HAL
open science

FrameSoC Workbench: Facilitating Trace Analysis through a Consistent User Interface

Generoso Pagano, Vania Marangozova-Martin

To cite this version:

Generoso Pagano, Vania Marangozova-Martin. FrameSoC Workbench: Facilitating Trace Analysis through a Consistent User Interface. [Technical Report] RT-0447, Inria. 2014, pp.26. hal-00977887

HAL Id: hal-00977887

https://inria.hal.science/hal-00977887

Submitted on 11 Apr 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

FrameSoC Workbench: Facilitating Trace Analysis through a Consistent User Interface

Generoso Pagano, Vania Marangozova-Martin

TECHNICAL REPORT
N° 447
March 2014
Project-Team MESCAL

ISSN 0249-0803 ISRN INRIA/RT--447--FR+ENG

FrameSoC Workbench: Facilitating Trace Analysis through a Consistent User Interface

Generoso Pagano *, Vania Marangozova-Martin †

Équipe-Projet MESCAL

Rapport technique n° 447 — March 2014 — 26 pages

Résumé : Les traces d'exécution sont un instrument précieux pour le débogage et l'évaluation de performances. Elles sont utilisées dans les cas d'applications complexes dans différents domaines, comme le calcul haute performance ou les systèmes embarques. Etant donné le volume croissant des traces, des outils puissants pour l'analyse de traces deviennent essentiels. FrameSoC, l'infrastructure de gestion de traces du projet SoC-Trace, fournit des solutions génériques pour simplifier l'analyse de traces [2,3]. Ce document décrit en détail l'environnement graphique fourni à l'utilisateur par FrameSoC. Nous décrivons les fonctionnalités utilisateur, ainsi que les mécanismes utilisés afin d'obtenir un environnement intuitif, extensible et configurable.

Mots-clés : Traces d'exécution, gestion de traces, infrastructure, représentation de données, interface utilisateur, interaction utilisateur, ergonomie, mécanisme publication/souscription, conception de logiciel.

This research is supported by the FUI SoC-TRACE project [1]

* INRIA, generoso.pagano@inria.fr
† UJF, Vania.Marangozova-Martin@imag.fr

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

FrameSoC Workbench: Facilitating Trace Analysis through a Consistent User Interface

Abstract: Execution traces are a precious instrument for debugging or performance evaluation of complex applications. They are used in multiple domains, including high performance systems and embedded systems. Given the constantly increasing trace size, powerful tools for trace analysis become essential. FrameSoC, the SoC-Trace project trace management infrastructure, provides generic solutions to facilitate trace analysis [2, 3]. This document describes the graphical environment provided by FrameSoC for execution trace analysis. We detail the functionalities provided to the user, as well as the mechanisms employed for the design of an intuitive, flexible and configurable environment.

Key-words: Execution traces, trace management, infrastructure, data representation, user interface, user interaction, ergonomics, publish-subscribe, software design.

Table of contents

1 Introduction **4**

2 FrameSoC Perspective **4**
2.1 Trace Browser 5
2.2 Trace Metadata Viewer/Editor 6
2.3 Event Density Chart 7
2.4 Statistics Pie Chart 8
2.5 Event Table View 9
2.6 Gantt Chart View 10
2.7 FrameSoC Menu and Toolbar 11
2.7.1 System Initialization 12
2.7.2 Tool Management 13
2.7.3 Color Management 14
2.7.4 Trace Import 15
2.7.5 Launching of an Analysis Tool 16
2.7.6 Trace Export 16

3 FrameSoC View Management Mechanisms **16**
3.1 Overview 16
3.2 Publish/Subscribe Inter-View Communication 17
3.3 Design of FrameSoC Views 19
3.4 Management of Multiple View Instances 20
3.5 View Correlation 21
3.6 Gantt Chart Architecture 22

4 Conclusions **24**

1 Introduction

Execution trace analysis, though useful for debugging and profiling complex applications, is often a very difficult task. Indeed, traces are far from standard: they may contain huge amounts of information, which differ in their formatting, encoding and semantics. In this context, trace management and analysis tools are crucial in helping the developer manipulate and analyze the data conveniently and efficiently. A critical point of these tools is the provided graphical user environment.

With FrameSoC, our trace management infrastructure, we provide a workbench that aims at fulfilling two main objectives. First, the workbench includes a set of functionalities to support and simplify common tasks performed on traces, from both management and analysis points of view. Second, the various functionalities are designed in order to provide a consistent graphical environment, where there is correlation among the different trace views.

This document describes the FrameSoC workbench, presenting both the user functionalities and the underlying design mechanisms. It is organized as follows. Section 2 provides a user-level description of the FrameSoC workbench: all the functionalities are described from the user's point of view, without providing implementation details. Section 3 provides a technical description of all the mechanisms supporting the FrameSoC workbench functionalities, with implementation details. Section 4 presents our conclusions and perspectives.

2 FrameSoC Perspective

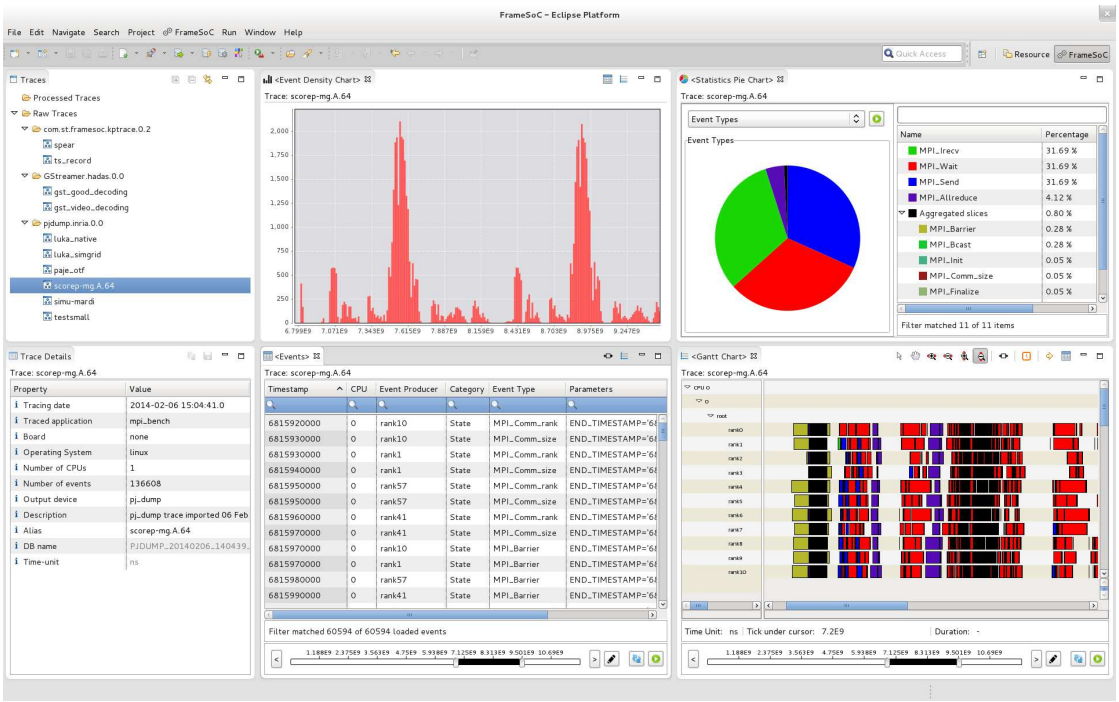


FIGURE 1 – FrameSoC Eclipse Workbench

The FrameSoC workbench is an Eclipse application providing a perspective [1] for trace management and analysis. The FrameSoC perspective (Figure 1) contains the following elements:

- A FrameSoC menu (on top).
- A toolbar (the second toolbar, from the left).
- A set of views for trace management and analysis. We find on the left the *management views*: a trace browser on top and a trace metadata viewer at the bottom. On the right there are the *analysis views*: an event density chart and a statistics pie chart on top, and an event table and a Gantt chart at the bottom.

The management views (trace browser and metadata viewer) refer to the whole system and are the entry point for trace analysis. Therefore they cannot be closed and there can be a single instance of each of them. On the contrary, all the analysis views refer to a single trace, so they can be opened and closed as needed. For each trace, there can be an instance of each type of analysis view. The maximum number of open instances for a given type of analysis view is configurable. When a trace is selected in the trace browser, the corresponding metadata are shown in the metadata viewer and all the analysis view referring to that trace (if any) are highlighted. Namely, the view name is surrounded by $<$ and $>$. In the example shown in Figure 1, all analysis views refer to the selected trace `scorep-mg.A.64`). On the other hand, when an analysis view is given focus, the trace shown in that view becomes the selected trace in the trace browser, so the metadata viewer is updated accordingly and all the analysis views are consistently highlighted or unhighlighted.

The following subsections describe the FrameSoC views, as well as the functionalities accessible via the FrameSoC menu and toolbar.

2.1 Trace Browser

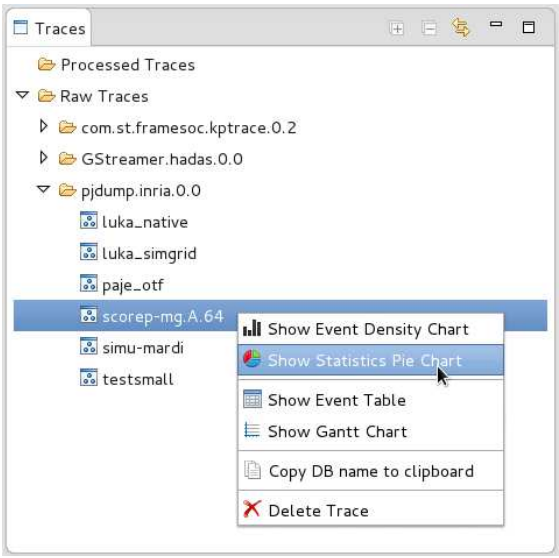


FIGURE 2 – Traces view

The *Traces* view (Figure 2) is the FrameSoC perspective trace browser. Traces are presented in a tree viewer with a two-level hierarchy. The first level distinguishes processed traces from raw

1. Within an Eclipse application, a perspective defines an initial set and layout of views, menu and toolbars

traces [2]. Then, in each of these two categories, traces are grouped by type. The type may relate to the trace format or to the tool that created the trace. The viewer presents a trace alias for each trace.

Right-clicking on a single trace gives access to the following functionalities:

- Show Event Density Chart: open the density chart view for this trace (see Subsection 2.3).
- Show Statistics Pie Chart: open the pie chart view for this trace (see Subsection 2.4).
- Show Event Table: open the event table view for this trace (see Subsection 2.5).
- Show Gantt Chart: open the Gantt chart view for this trace (see Subsection 2.6).
- Copy DB name to clipboard: copy to the clipboard the name of the database containing the trace raw data.
- Delete Trace: delete the trace from the system.

If more traces are selected, only the *Delete Trace* entry is displayed in this context menu. Note that if one or more traces are deleted, an update notification is sent to the other views. This allows a view to do the necessary actions if the trace it displays has been deleted.

The view toolbar contains three buttons: a couple of buttons to expand/collapse the trace hierarchy and a button to manually resynchronize the displayed traces with the information contained in the FrameSoC System DB (see the technical report RT-427 [3] for further details on the database architecture).

2.2 Trace Metadata Viewer/Editor

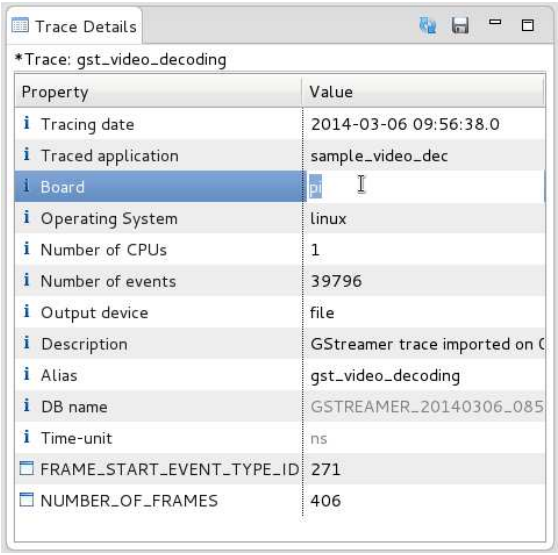


FIGURE 3 – Trace Details view

The *Trace Details* view (Figure 3 is the FrameSoC perspective trace metadata viewer and editor. When a trace is open or selected in the *Traces* view, the trace metadata are displayed in a table viewer containing two columns: the property name and the property value. The different properties are grouped in two different categories: predefined properties (displayed first) and custom properties (displayed last). Those categories are identified by two different icons. The

2. As described in the technical report RT-427 [3], a processed trace is a trace created by an analysis tool as the result of an analysis on another trace.

Value column is normally editable, with the exception of some properties that are read only (the database name and the time unit). In order to edit an editable property value, you simply have to click the value and modify it (as shown in Figure 3 for the *Board* property). When one or more values have been modified, a star (*) is displayed before the trace alias, on top of the table, and the two view toolbar buttons are enabled. The *reset* button restores all non-saved edited properties to their previous value. The *save* button stores the changes. If the *Alias* predefined property of a trace is persistently modified, other views are notified in order to take the necessary actions (e.g. update their label for the trace).

Note that if more than one trace is selected in the *Traces* view, the metadata view displays only the properties having the same name and the same value. For these properties the editing support is still working, with the note that all the selected traces metadata are modified.

2.3 Event Density Chart

FIGURE 4 – Event Density Chart view

The *Event Density Chart* view (Figure 4 is a FrameSoC analysis view displaying the event density over time in the form of a histogram. The x axis represents the time, while the y axis represents the number of events. The histogram number of bins is fixed and has been chosen in order to ensure a clear visualization, taking into account the number of pixels actually present on a screen. The user can zoom and dezoom portions of the chart, using the mouse as shown in Figure 4. In particular, to zoom a portion of the chart, the user has to click to the start of the portion, drag the mouse going to the right up to the end of the portion, then release the click. To completely dezoom, the user has to do the same as above, but dragging the mouse to the left this time. Hovering the mouse on a bin, some information about the bin is displayed (the central timestamp and the number of events).

The view toolbar contains two buttons, a *table* and a *gantt* button, which trigger the visualization of the displayed portion of trace in the table view and in the Gantt view respectively.

Note that, for the computation of the event density, all the events of the trace are considered. Among the possible improvements of this view, we foresee the possibility to chose which type of events must be considered in the computation. Going even further, it would be interesting to reuse the histogram representation to visualize different metrics, as for example the probability density function of the duration of a given type of state in the trace [3].

3. For the concept of *state* in a trace (and other high level trace entities) see the original Pajéformalization [4] and the adaptation in FrameSoC [5]

2.4 Statistics Pie Chart

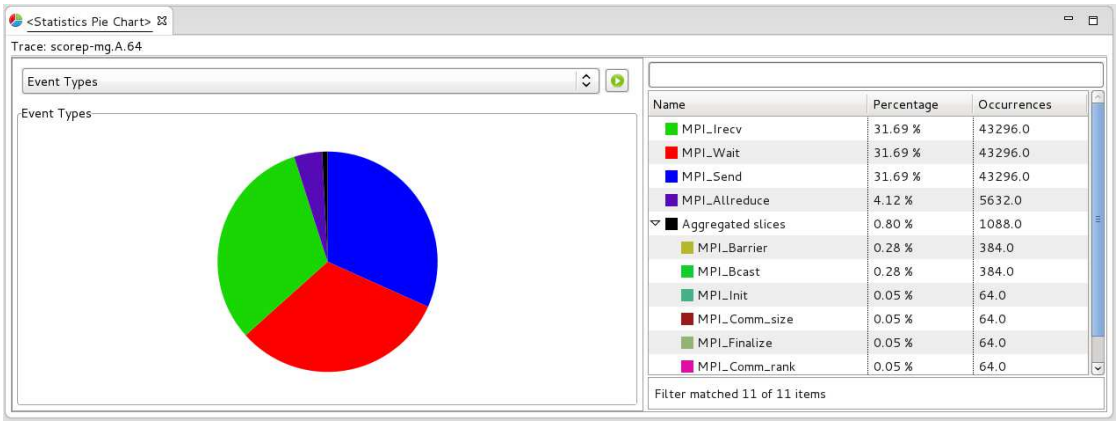


FIGURE 5 – Statistics Pie Chart view

The *Statistics Pie Chart* view (Figure 5 is a FrameSoC analysis view presenting several metrics in the form of a pie chart. The view is composed of two parts. On the top left, there are the metric selector and a *play* button to load the metric. At the bottom left, there is the actual pie chart. On the right, a table viewer displays the same information as the pie. This table has three columns: the pie-slice name, the percentage value and the actual value. The name column cells contain a small square icon, filled with the color corresponding to the slice. Each column header, when clicked, triggers the sorting of the rows according to the values in the corresponding column. On the top of this table, there is an editable field, acting as a filter on table rows, working on the first column (*Name*). The number of items matched by the filter, over the totality of items, is shown in the status bar under the table viewer.

For the moment being, two metrics are available: the *Event Producer* instances and the *Event Type* instances. Each slice represents the number of events having a given event producer or a given type respectively.

From Figure 5 it is possible to note that there can be a special slice in the pie: the *Aggregated slice*. This slice aggregates all the slices whose value is smaller than a given threshold, being therefore difficult or impossible to see. This threshold has been empirically set to 1%, taking into account user ergonomics and screen limitations. In the table on the right, the *Aggregated slice* corresponds to a folder entry, whose sub-entries are the actual slices. All the detailed information is thus kept and available in the tabular representation.

Note that when a *Statistics Pie Chart* view is opened for a given trace using the context menu in the trace browser, no pie chart is actually displayed, since the user has to select the metric of interest first, then press the *play* button.

For the future, we plan to improve this view adding more metrics, as for example the duration of the various states [5] or the duration of the active time of each producer. For these new metrics and for the existing ones, the possibility to use only given event producers or event types in the statistics computation would be useful (e.g., create the *Event Type* pie chart using only the event types A, B and C). We also foresee the possibility to compute the different metrics considering only the events in a given time interval (and not always the whole trace). Coming to aggregation, it would be useful to let the user decide how to aggregate, interactively defining custom aggregated slices (e.g., I want to create an aggregated slice X, containing the actual slices a, b and c).

2.5 Event Table View

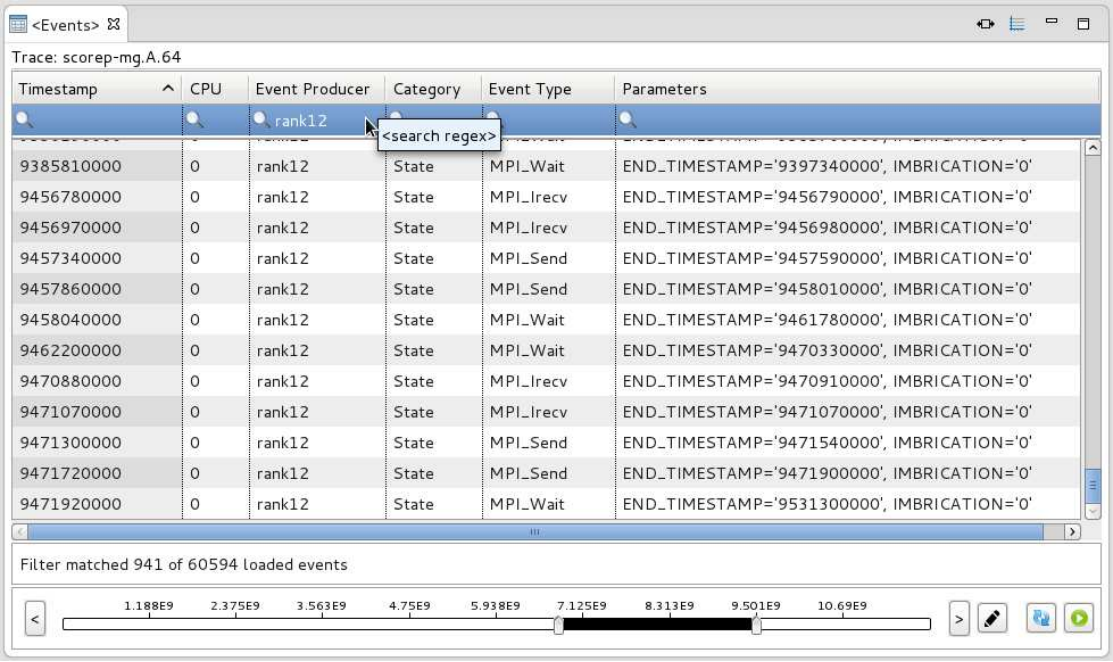


FIGURE 6 – Events view

The *Events* view (Figure 6 is a FrameSoC analysis view showing a tabular representation of trace events. The main element of this view is a table viewer, displaying a distinct row for each event of the trace. This table has the following columns:

- Timestamp: the event timestamp.
- CPU: the number of the CPU on which the event has been produced.
- Event Producer: the name of the entity producing the event.
- Category: the event category (*Punctual Event*, *State*, *Link* or *Variable* [5]).
- Event Type: the event type name.
- Parameters: the list of the event custom parameters, with the format NAME='VALUE'. For the events whose category is not *Punctual Event*, this list is preceded by the two parameters that qualify a given category [5], always presented with the same format.

Each column header, when clicked, triggers the sorting of the rows according to the corresponding column. The first row of the table contains an editable filter for each column. These filters accept regular expressions. For example, in Figure 6 only the events produced by a given producer are filtered. The number of events matched by the filter, over the totality of loaded events, is shown in the status bar under the table viewer.

At the bottom of the view, there is a time management bar. This bar contains a double range time slider (representing the whole trace duration) surrounded by two arrow buttons, and three more buttons on the right (*edit*, *reset* and *play* buttons). The two knobs of the double range slider identify the portion of the trace (colored in black) actually loaded in the table. This way, the user always keeps a global visibility on the whole trace, while loading only the information he is interested in. In order to change the time window loaded in the table, one needs to change the width of the black bar. However, in order to avoid spurious and useless data transfers (from

disk to memory), the change is executed only when the user activates the *play* button on the right of the bar. The *reset* button resynchronizes the time bar with the time window actually loaded in the table, if they differ. In order to modify the time window visualized in the double range slider the user has several possibilities.

- Graphical selection: it is the most intuitive way and it involves using the two knobs, in order to graphically set the two bounds of the time interval.
- Time window navigation: this possibility involves using the two arrow buttons placed respectively on the left and on the right of the time bar, in order to select a time window that has the same size of the currently visualized one, but is located immediately before or immediately after the current one.
- Manual selection: by pressing the *edit* button placed immediately beside the time bar on the right, the user access a dialog (Figure 7) where it is possible to explicitly put the exact values of the time interval start and end timestamps. This dialog enables also a manual definition of the size of the time window to be used when performing the time window navigation described above.

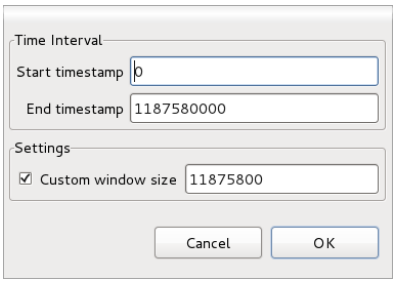


FIGURE 7 – Time window dialog

The view toolbar contains two buttons. The first one (*adjust*) triggers column width resizing in order to fit actual content. The second one (*gantt*) triggers the visualization of the current loaded time window in the Gantt view.

2.6 Gantt Chart View

This view (Figure 8 shows a Gantt chart representation of trace events. This kind of representation is pretty common in trace analysis frameworks (e.g., Lttng Eclipse Viewer [6]) and is classically used to visualize application behavior over time, thanks to its ability to represent causality relations [7]. The main element of this view is a Gantt viewer [4], which is composed of two parts. On the left there is the hierarchy of event producers, grouped by CPU. On the right there is the actual time chart, showing for each producer all the punctual events it generates (represented as simple vertical lines) and all the states it spends time in (represented as colored rectangles). If there are links in the trace (e.g., communications), they are represented as oriented arrows. Note that the colors used to fill the rectangles depend on the event type. For a given event type, the same color is used in this Gantt view and in the Pie Chart view (Subsection 2.4). Note that hovering the mouse on a given state, triggers the displaying of a tooltip showing the event type name (Figure 8).

The view toolbar contains the following buttons:

- Six buttons enable six different modes, where a mode defines the behavior of a mouse click on the time chart:

4. The viewer is part of a visualization library, provided by STMicroelectronics [8].

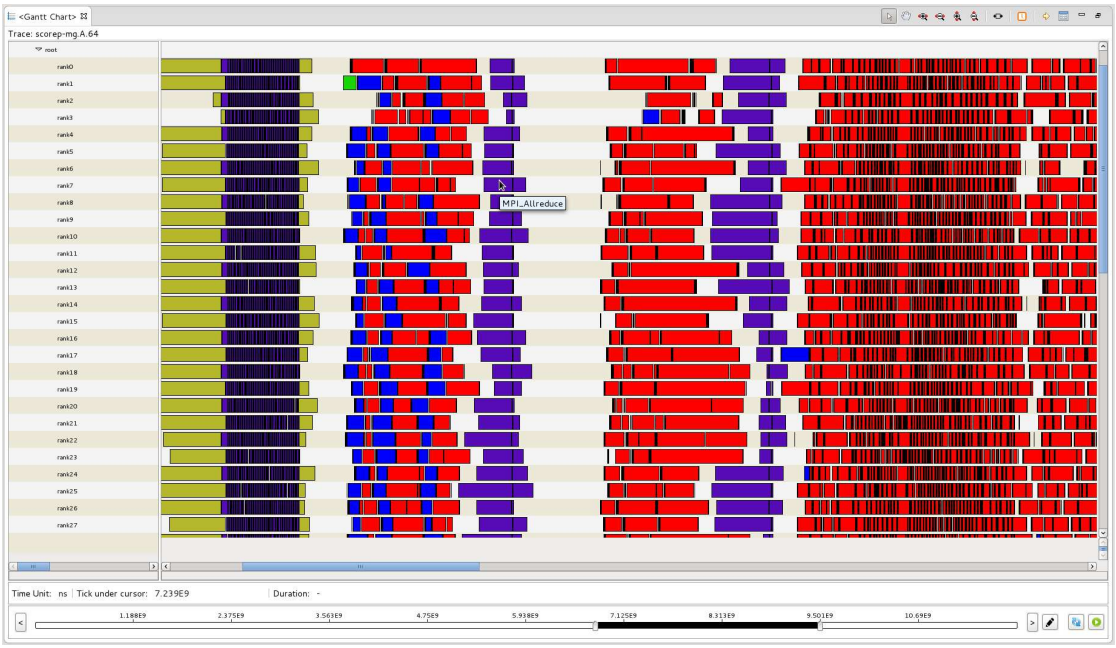


FIGURE 8 – Gantt Chart view

- Select mode: it is possible to select with the mouse a time region in the time chart.
- Drag mode: it is possible to drag the time chart with the mouse.
- Horizontal Zoom In
- Horizontal Zoom Out
- Vertical Zoom In
- Vertical Zoom Out

- The *adjust* button triggers the resize of content to fit the view size.
- The *clock* button enables/disables the time compression feature: if enabled, this feature hides large time regions without events.
- The *show all* button triggers the loading of the whole trace in the Gantt.
- The *table* button triggers the visualization of the current loaded time window in the Event Table view.

Under the Gantt viewer, there is a status bar displaying the trace time unit, the timestamp currently under the mouse cursor and the selected interval duration, if any.

Under the status bar, at the bottom of the view, there is a time management bar, which has exactly the same behavior as the one in the Event Table view (see Subsection 2.5 for the details). As it happens for the table, a new time window is loaded in the Gantt viewer only on demand, when the *play* button is pressed. Note that this time, given that it is possible to select a time region also on the Gantt viewer (if in selection mode), the selections in the viewer and in the double range slider are always synchronized, thus helping trace navigation.

2.7 FrameSoC Menu and Toolbar

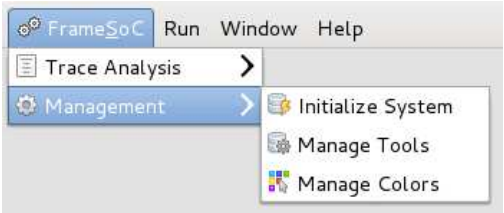
When the FrameSoC perspective is activated, the FrameSoC menu (Figure 9) and the corresponding toolbar (Figure 10) are visible.

In the menu, the different functionalities are grouped in two categories: management and trace

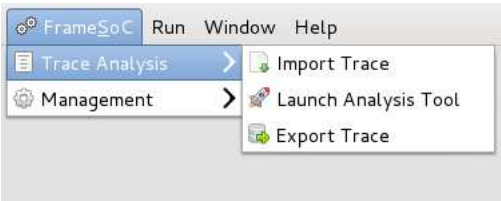
analysis. The management menu (Figure 9a) allows the user to access system configuration, tool management and color management. The trace analysis menu (Figure 9b) allows the user to launch importers, analysis tools and exporter tools.

In the toolbar, the different buttons are simply shortcuts for the above functionalities, where corresponding icons mean corresponding functionalities. The added value of the toolbar is that, beside each of the three buttons used to launch the tools of the various category (import, analysis, export), there is a drop down menu containing the list of the tools of that category, useful to directly launch a tool.

In the following subsections, the functionalities accessible from the menu or the toolbar are explained in detail.



(a) Management Menu



(b) Trace Analysis Menu

FIGURE 9 – FrameSoC Menu

FIGURE 10 – FrameSoC Toolbar

2.7.1 System Initialization

At system initialization, the user accesses a configuration wizard, whose first page is shown in Figure 11. This page allows the user to select the DBMS [5] to be used for trace storage. In fact, as described in the technical report RT-435 [9], FrameSoC can work with several DBMS. At the moment being, the support has been implemented for SQLite (recommended) and MySQL.

Once the user has chosen the DBMS and pressed *Next*, he comes to the DBMS configuration page (Figure 12), which is different for each DBMS. If SQLite is selected (Figure 12a), the user has simply to enter the directory where he wants the database files to be kept. Otherwise, if MySQL is chosen (Figure 12b), the user has to specify some connection parameters (user-name, password, URL).

In both cases, after pressing *Finish*, the FrameSoC storage subsystem is correctly configured. If a System DB already exists, it is reused, otherwise a new one is created. This configuration is saved in the FrameSoC configuration file (~/.soctrace.conf).

Note that after the initialization, in the case of existing System DB, if there is a mismatch between the tools registered in this System DB and the tools actually present in the FrameSoC Eclipse runtime [6], this is automatically fixed:

5. Data Base Management System

6. As described in the technical report RT-435 [9] the preferred way to add tools to FrameSoC is to create an Eclipse plugin, extending a specific extension point defined by FrameSoC. For this reason a tool is typically a plugin in the FrameSoC Eclipse runtime.

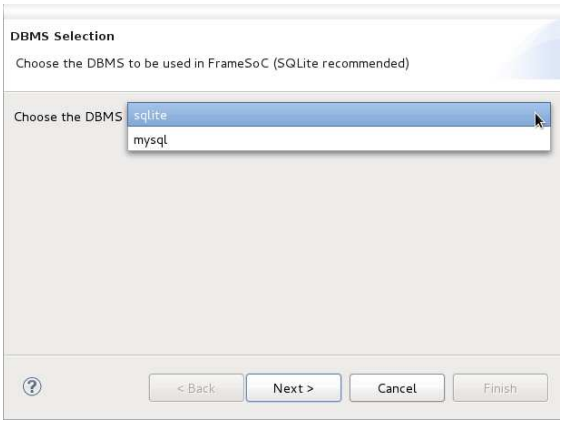
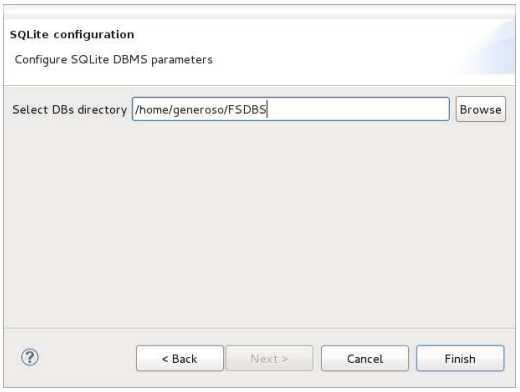
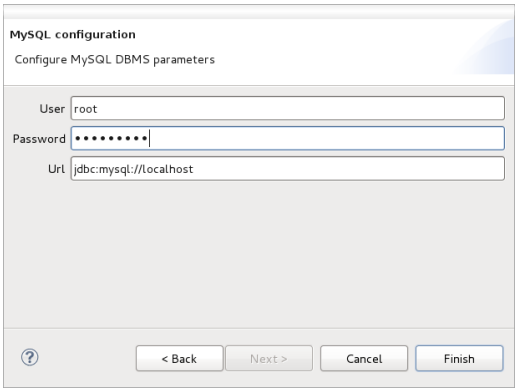


FIGURE 11 – System Initialization: DBMS selection dialog



(a) SQLite configuration dialog



(b) MySQL configuration dialog

FIGURE 12 – System Initialization: DBMS configuration

- If a tool exists in the DB but not in the runtime, the tool is removed with its results (if any), after user confirmation.
- If a tool exists in the runtime but not in the DB, the tool is automatically registered.

Note also that at each FrameSoC startup, the system automatically checks that the storage configuration is good. If it is not the case, the system initialization wizard is automatically launched. A control for mismatch between runtime tools and System DB tools is equally done at each startup, with the same policy as described above.

2.7.2 Tool Management

The tools management dialog (Figure 13) simply displays the list of tools registered to the system, enabling the installation, modification and removal of external *black-box* tools (tools that are not eclipse plugins).

Adding or modifying a *black-box* tool requires the user to edit the different fields of the dialog displayed in Figure 14. In particular, the user has to pick a unique tool name, specify the launching command, select the tool type (import, analysis, export), write the launching documentation.

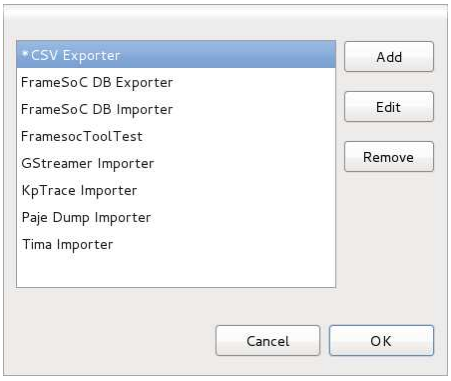


FIGURE 13 – Tool manager

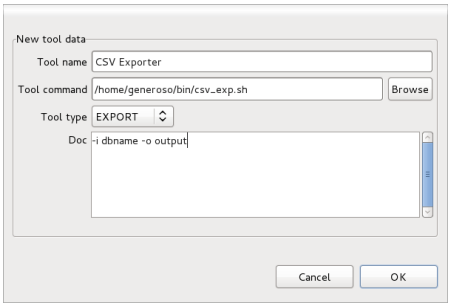


FIGURE 14 – Black-box tool dialog

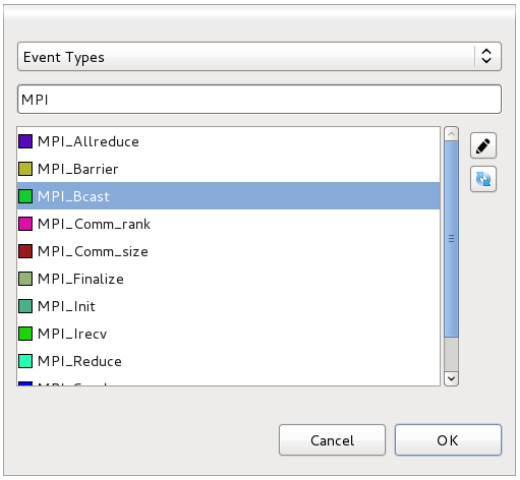
Note that the use of external *black-box* tools is discouraged, since most of the features available using Eclipse plugins are not available.

Note also that plugin tools cannot be added, edited or removed via the manage tool dialog, since they are managed as standard Eclipse plugins. Plugin tools are installed and removed using the normal Eclipse procedure (Help -> Install New Software...). Furthermore, the extension point for tools (described in the technical report RT-435 [9]) defines the plugin tool metadata and launching class.

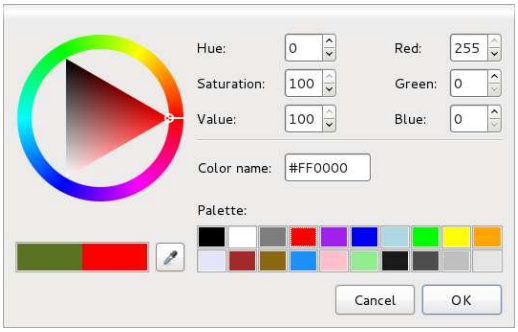
2.7.3 Color Management

The color management dialog (Figure 15a) allows the user to modify the colors associated to event producers and event types in a centralized way for the whole workbench. The combo box at the top of the dialog enables the selection of the entity (event producer or event type). The list below enumerates all the entities, preceded by a small squared icon filled with the entity color. The editable text field on top of this list can be used as a filter on the list. When an entity is selected, pressing the *edit* button gives access to the color edition dialog (Figure 15b). Pressing *OK* makes all changes to be saved. Pressing the *reset* button (under the *edit* button), reverts unsaved changes. The color configuration for a given entity is physically stored in a configuration file located in the `configuration/fr.inria.soctrace.framesoc.ui` sub-folder of the eclipse install directory. For example, the relative path to eclipse install directory of the event type color configuration file is: `configuration/fr.inria.soctrace.framesoc.ui/event_type_colors`.

Note that when OK is pressed after changing some colors, the workbench modules are notified



(a) Color management dialog



(b) Color editing dialog

FIGURE 15 – FrameSoC Color Management

and the views may react by updating their colors in real-time. In FrameSoC, both the Pie Chart and the Gantt Chart views implement this functionality.

2.7.4 Trace Import

The trace import dialog (Figure 16) allows the user to import a new trace into the system using one of the registered importer tools. The user selects the importer from the combo box at the top, then he specifies the trace files (if more than one file is needed, all the files should be selected in the browser dialog opened when the *Browse* button is pressed). If the importer requires additional parameters (the *Doc* field normally provides this information), the user specifies these parameters too.

Finally, after pressing *OK*, the import process is launched. At the end of this process, the trace browser view is automatically updated with the new trace information.

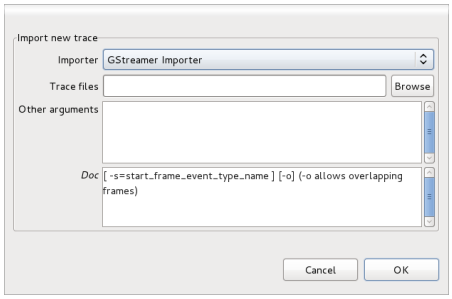


FIGURE 16 – Import trace dialog

2.7.5 Launching of an Analysis Tool

The launch analysis tool dialog (Figure 17) allows the user to launch one of the analysis tools registered to the system. The user selects the analysis tool from the combo box at the top. If the tool requires additional parameters (the *Doc* field normally provides this information), the user specifies these parameters too. Finally, after pressing *OK*, the tool is launched.

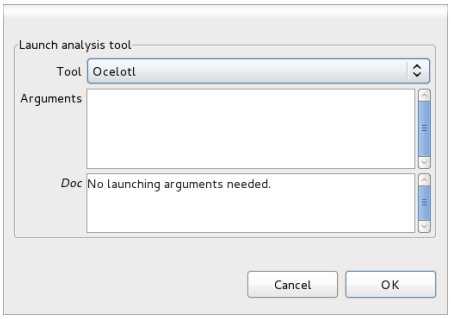


FIGURE 17 – Launch analysis dialog

2.7.6 Trace Export

The trace export dialog (Figure 18) works exactly as the launch analysis tool dialog, with the difference that this time the user can launch one of the exporter tools registered to the system.

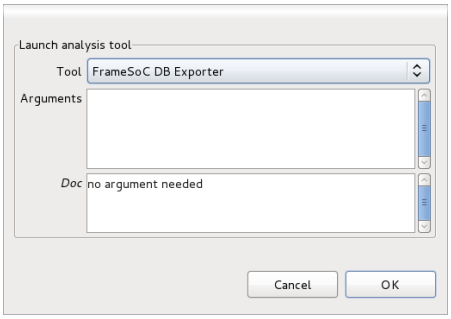


FIGURE 18 – Export trace dialog

3 FrameSoC View Management Mechanisms

3.1 Overview

The FrameSoC workbench has pursued several goals concerning user interactions, multiple-view target behavior and development facilities.

From the *user interaction perspective*, our workbench focuses on a set of views which are intuitive to use and which support simple interactions for trace data analysis. The set of views is designed to be *open* in the sense that it is not predefined and allows for the integration of new types of views. The set is also *configurable* as it is possible to choose, among the available views,

which ones to use in the working environment. A useful feature to take into account, especially when working with multiple traces, is the possibility to have multiple instances per type of view.

From the *view correlation perspective*, all views need to show consistent data representations. They should be able to represent the same trace data, during the same time interval and with the same color code. Moreover, view updates need to be correlated: when a user interacts with a view, the executed changes are consistently reflected in other views.

From the *development perspective*, our workbench facilitates the engineering and the integration of new types of view. It defines a global management architecture and provides predefined classes and interfaces to ensure the proper integration of newly developed views.

Figure 19 provides a high level vision of our workbench software architecture. The workbench is based on the design facilities provided by the Eclipse framework. The managed graphical views are provided as a packaged software module containing a set of Eclipse plugins. Views are managed through the Eclipse extension and extension point mechanism. In the Eclipse environment, extension points define the contracts between modules and are used for loose coupling. They may represent configuration information only or some code contribution. Extensions provide implementations for these contracts. The *management views* provided by our framework (the trace browser and the metadata viewer/editor) are managed as standard Eclipse views. On the contrary, the contribution of new *analysis views* for our environment is facilitated by our definition of a particular extension point. During a working session, the views implementing this specific extension point are managed using two mechanisms. At a lower level, we use a Publish-Subscribe architecture for flexible and scalable communication. Above this layer, we implement control entities to ensure consistent view correlation and take care of some performance issues.

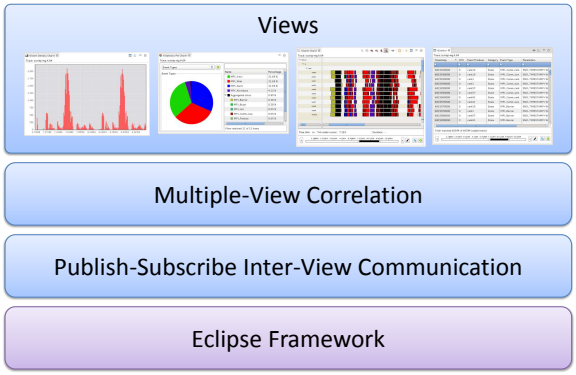


FIGURE 19 – Framesoc Workbench Software Architecture

3.2 Publish/Subscribe Inter-View Communication

In order to support the correlation among views and achieve a global consistency in the analysis environment, we designed and implemented a simple communication mechanism based on the Publish-Subscribe messaging pattern [10]. The Publish-Subscribe architecture is based on the idea of decoupling message senders, called publishers, from message receivers, called subscribers. Messages are grouped in *topics*, so publishers can publish data for a given topic, without knowing who and how many the receivers of these data (if any) are. On the other hand, subscribers can subscribe to one or more topics, thus receiving all the data published on those

topics, without knowing the publishers of these data. This architecture achieves separation among modules, allowing for scalable growth with small development effort.

The FrameSoC Publish-Subscribe architecture is called FrameSoC Bus. The three main functionalities available on this bus are:

- Send a message for a given topic.
- Register to a given topic.
- Unregister from a given topic.

The bus is physically implemented by the *FramesocBus* singleton, which offers the necessary *send(topic, data)*, *register(topic)*, *unregister(topic)* methods. Subscribers modules must implement the *IFramesocBusListener* interface, which provides the *handle(topic, data)* method. Message sending is synchronous, meaning that when a message is sent, all the handle methods of the subscribers are called sequentially. For this reason, the actions done in the handle methods should not be time consuming: if long operations are needed, they should be done in a different thread.

The Publish-Subscribe approach offers several advantages compared with a traditional *Listener* approach (as used for example in SWT [11]). First, it does not require the sender and the receiver to know each other: on the contrary, in order to add a listener to a given object, the reference to this *sender* object is needed. Second, the Publish-Subscribe approach makes it easy an incremental growth of the system: to create a new category of messages/events it is enough to create a new *topic* and send data for it, without the need of providing new interfaces or base classes for listeners and events.

In the FrameSoC Bus the following topics are currently used:

`TOPIC_UI_FOCUSED_TRACE`
: Messages for this topic are sent when a FrameSoC analysis view gets focus. The subscribers are currently the trace browser and the trace metadata viewer/editor. The message body is the Trace object corresponding to the trace shown in the focused view.

`TOPIC_UI_SYNCH_TRACES_NEEDED`
: Messages for this topic are sent when the information in the trace browser must be reloaded from the FrameSoC System DB. The message body is a boolean stating if the event must be processed or not by other modules. The typical usage of this topic is to notify the analysis environment that a new trace has been added to the system or that a trace has been deleted.

`TOPIC_UI_TRACES_SYNCHRONIZED`
: Messages for this topic are sent when the trace browser has been synchronized with the DB. The message body is a map of three trace lists: new traces, deleted traces, updated traces.

`TOPIC_UI_REFRESH_TRACES_NEEDED`
: Messages for this topic are sent when a view displaying metadata about traces must be simply refreshed (DB is not used) using the current trace objects stored in memory. The message body is `null`. This topic is used for example when trace metadata are changed by the trace metadata viewer: the trace objects in memory are already updated, so the UI can simply reload them, without having to read the DB.

`TOPIC_UI_SYSTEM_INITIALIZED`
: Messages for this topic are sent after the system has been initialized. The message body is `null`.

`TOPIC_UI_TABLE_DISPLAY_TIME_INTERVAL`
: Messages for this topic are sent when we want to display a time interval for a given trace

in the Table view. The message body is a `TraceIntervalDescriptor`, which contains the Trace object and the time interval bounds.

`TOPIC_UI_GANTT_DISPLAY_TIME_INTERVAL`
: Messages for this topic are sent when we want to display a time interval for a given trace in the Gantt view. The message body is a `TraceIntervalDescriptor`, as above.

`TOPIC_UI_COLORS_CHANGED`
: Messages for this topic are sent by the color management dialog when the user saves a new set of colors. The message body is a `ColorsChangeDescriptor`, which simply specify if the change affects event types or event producers.

The FrameSoC Bus, through the *FramesocBus* singleton, provides also a simple service for sharing variables. Basically the singleton provides methods for:

- Setting/updating a shared variable, specifying the name and the value.
- Reading a shared variable, specifying the name.

Both operations are atomic. On the contrary, the atomicity of a sequence *read-test-write* on a variable must be ensured by the user code. The need behind the use of this shared variable mechanism is the following: a module joins the system (e.g., a new FrameSoC analysis view is opened) and needs to know some information about an event that happened before its arrival (e.g., the trace being selected in the FrameSoC Trace Browser). The relevant information related to this event are stored in a shared variable when the event occurs, and then retrieved after, when needed.

Currently two variables are defined in the FrameSoC Bus:

`TRACE_VIEW_SELECTED_TRACE`
: Trace object corresponding to the selected trace in the trace browser. If more than one trace is selected, this variable stores the first selected trace.

`TRACE_VIEW_CURRENT_TRACE_SELECTION`
: Eclipse `ISelectionObject` object corresponding to the current selection in the trace browser. If more traces are selected, they all appear within this object.

3.3 Design of FrameSoC Views

As stated in the user level description proposed in Section 2, the FrameSoC perspective contains two different kind of views:

- Management views: the Trace Browser (Subsection 2.1), and the Trace Metadata Viewer/Editor (Subsection 2.2).
- Trace analysis views: the Event Density Chart (Subsection 2.3), the Statistics Pie Chart (Subsection 2.4), the Event Table View (Subsection 2.5), and the Gantt Chart View (Subsection 2.6).

The views of the first group refer to the whole system, so they are unique instances and cannot be closed. The views of the second group refer to a single trace, so there can be an instance of a given type of view for each trace [7] and it is possible to close such instances. These views have to implement a consistent behavior regarding the trace they are showing (e.g. title highlighting when the trace is selected).

FrameSoC management views simply extend the predefined Eclipse `ViewPart` base class. They are plugged to the FrameSoC perspective using the predefined Eclipse extension point for views (`org.eclipse.ui.views`). FrameSoC analysis views, however, ex-

7. For analysis views, a parameter in the FrameSoC configuration file (∼/.soctrace.conf) defines the maximum number of instances for a given type of view.

tend the `FramesocPart` abstract class, which provides the implementation of common basic functionalities and defines the interface for specific behaviors. External Eclipse plug-ins may contribute FrameSoC analysis views via a specifically designed extension point (`fr.inria.soctrace.framesoc.ui.perspective.part`), provided by the FrameSoC user interface plugin. This extension point enhances modularity and simplifies the creation of new analysis views, by giving access to all the information needed by FrameSoC to correctly manage an analysis view. Indeed, it specifies creation details, perspective positioning, updates of FrameSoC contextual menus, as well as execution commands.

More in detail, this extension point defines the following fields:

- `viewId`
  It is the ID of the analysis view provided. This view must extend FramesocPart.
- `icon`
  Framesoc analysis view icon, to be displayed in the trace browser context menu.
- `launchCommand`
  ID of the command [8] opening this FrameSoC analysis view. The command name is used to display a text in the trace browser context menu.
- `position`
  Position of the analysis view in the FrameSoC perspective. It may be one of `TOP_RIGHT` and `BOTTOM_RIGHT`.
- `priority`
  Integer representing the analysis view priority regarding the positioning in the perspective and in the trace browser context menu, within a given position (as defined above). A small integer means a higher priority, that is the view is presented first: more at left in the FrameSoC perspective, more on top in the context menu.

The class `FramesocPartContributionManager` manages this extension point, providing a simple interface to get all the extensions (the actual analysis views) actually present in the current Eclipse runtime.

Note that this extension point requires only a view ID and a command ID, but the plugin extending this extension point has to effectively provide an extension for the extension points `org.eclipse.ui.views` and `org.eclipse.ui.commands` for those IDs.

Note also that FrameSoC predefines some view IDs, to be used for analysis views wanting to offer a service known to the system. These constants are defined in the class `FramesocViews`. For example, a view able to represent a trace in the form of a Gantt will have an ID equal to `FramesocViews.GANTT_CHART_VIEW_ID` in order to notify to FrameSoC that there is a Gantt in the system. This enables easy management of predefined functionalities (like the "show in Gantt" or "show in table" buttons), though not preventing the contribution of FrameSoC analysis views with custom IDs, but still respecting the extension point and extending the `FramesocPart` class.

3.4 Management of Multiple View Instances

As specified in Subsection 3.3, FrameSoC analysis views may exist in multiple instances (one instance of each type of analysis view per trace). Within the Eclipse framework, each type of view has a unique identifier. If we want to have more instances for a given type of view, we have to provide a unique *secondary* identifier for that type of view. As for version 4.3.2, the Eclipse framework does not automatically set a valid secondary ID (it is `null`) when opening a view through the *Show View* menu. Therefore, to correctly manage unique secondary identifiers for a given type of view, we use two classes: the `FramesocPartManager` singleton, and the `FramesocPart` abstract class.

8. We refer here to the concept of *command* as defined in the Eclipse framework.

The unique identifiers are assigned by the `FramesocPartManager` singleton, which is in charge of creating new FrameSoC analysis views. Indeed, this singleton provides a method to instantiate a new `FramesocPart` instance, given its ID [9]. The prototype of this method is: `getPartInstance(String viewID, Trace trace)`, where `viewID` is the ID of the FrameSoC analysis view to create and `trace` is the trace that must be visualized in the view. The method, first of all, checks if the given trace is already loaded in an analysis view of the given type [10]. If it is the case, the existing view is simply given focus. Otherwise the method checks if an empty view for the given type exists: in such a case, the empty view is used to load the trace. If none of these conditions is satisfied, a new view instance for the given type must be created (if the maximum number of instances for a type has not been reached yet), assigning a new unique secondary identifier. Given that Eclipse stores in a workbench configuration file [11] the previously used secondary IDs, the `FramesocPartManager` singleton has to ensure secondary ID uniqueness also considering past used IDs. A simple and effective way to do that is to use the method `UUID.randomUUID().toString()`, which generates a string corresponding to the hexadecimal representation of a 128 bit a random value, to create the secondary ID.

The base `FramesocPart` class, on the other hand, ensures that, even if the view instance is opened through the Eclipse *Show View* menu (and not via the FrameSoC context menu in the trace browser), it acquires a valid secondary ID. It does that simply intercepting in its constructor the *standard* Eclipse view creation, aborting it, and completing the operation via the `FramesocPartManager` singleton `getPartInstance()` method.

3.5 View Correlation

The `FramesocPart` class offers support for a common behavior regarding the consistent management of the trace currently visualized by the analysis view. In particular, the following functionalities are provided:

- When the view is given focus, if the trace currently stored in the `TRACE_VIEW_SELECTED_TRACE` variable differs from the trace visualized in the view, the variable is updated with this trace and an message is sent on the FrameSoC Bus for the topic `TOPIC_UI_FOCUSED_TRACE`, to notify the other views about the new currently selected trace.
- The view can be highlighted and unhighligted (as a reaction to the change of the currently selected trace). In the current implementation, highlighting means simply surrounding the view name by $<$ and $>$. A different solution (as putting in bold the view name), would be easily inherited by all FrameSoC analysis view, simply modifying the base class implementation.
- The view is a `IFramesocBusListener` and provides basic handling mechanisms for the events related to trace alias change, trace removal or generic trace metadata update (`TOPIC_UI_REFRESH_TRACES_NEEDED` and `TOPIC_UI_TRACES_SYNCHRONIZED` topics). In particular, when the trace alias changes, the content description of the trace is updated, and when the trace is deleted, the view is hidden (disposed).
- The view, being designed to be a `IFramesocBusListener`, also ensures automatic unregistering from followed topics at view disposal.

Moreover, the `FramesocPart` class defines the interface for the method used by the framework to display a trace in the analysis view. The prototype of this method is: `showTrace(Trace trace,`

9. It must be an ID of a view extending `FramesocPart` and advertised in the extension point `fr.inria.soctrace.framesoc.ui.perspective.part` described in Subsection 3.3.
10. Remember that the primary ID of the view represents the type of view
11. ./runtime-EclipseApplication/.metadata/.plugins/org.eclipse.e4.workbench/workbench.xmi

`Object data)`, where `trace` is the trace to be visualized and `data` is an object that depends on the view type and is used to give some instructions about how the trace should be visualized (for example, for the Event Table View this object is a `TraceIntervalDescriptor`, which specifies what time interval should be visualized in the table). If `data` is `null`, the view should implement a default behavior in visualizing the trace (for example, still in the case of the Event Table view, all the trace will be visualized).

The `FramesocPart` class works in symbiosis with the `FramesocPartManager` singleton, which centralizes all management operations on FrameSoC analysis views, in order to have the necessary consistency in the whole environment.

In order to centrally coordinate the FrameSoC analysis view behavior, the `FramesocPartManager` singleton is a `IFramesocBusListener`, which observes all the topics related to environment consistency and view correlation. In particular, in its `handle()` method it manages the following topics:

- `TOPIC_UI_GANTT_DISPLAY_TIME_INTERVAL`: if the `FramesocPartContributionManager` says that there is a plugin providing a Gantt view via the `fr.inria.soctrace.framesoc.ui.perspective.part` extension point, such a view is created following the policy described above, and its `showTrace()` method is called, specifying the required time interval.
- `TOPIC_UI_TABLE_DISPLAY_TIME_INTERVAL`: if the `FramesocPartContributionManager` says that there is a plugin providing a Table view via the `fr.inria.soctrace.framesoc.ui.perspective.part` extension point, such a view is created following the policy described above, and its `showTrace()` method is called, specifying the required time interval.
- `TOPIC_UI_FOCUSED_TRACE`: when a new trace is selected in the workbench, all the analysis views displaying this trace are highlighted and all the other views are unhighlighted [12].

The `FramesocPartManager` singleton provides also some utility functions for FrameSoC analysis view management, used in other modules. In particular:

- `cleanFramesocViews()`: reset the FrameSoC perspective, leaving only one empty instance for each analysis view.
- `disposeView(FramesocPart part)`: dispose a FrameSoC analysis view, updating the instance counter.
- `isViewExisting(String id)`: check if a given type of view exists in the runtime.
- `updateTitlesHighlight(Trace trace)`: highlight/unhighlight views when the given trace is selected.

3.6 Gantt Chart Architecture

As described in Subsection 2.6, the Gantt chart visualization is especially critical for trace analysis, since it helps the understanding of trace behavior by highlighting causal relations among events. Given the central role of the Gantt and given that FrameSoC aims at being a generic framework for trace analysis, the Gantt Chart View provided [13] has been designed in order to answer to the need of easily customizing trace visualization. Different trace formats, in fact, may have different ways to represents events on a time line: they may require special icons, customized texts, different types of links, or, going even further, they may require a specific processing over

12. Note that the same occurs when the *Eclipse Selection Service* (https://www.eclipse.org/articles/Article-WorkbenchSelections/article.html) notifies the workbench that a new selection has been done in the trace browser tree viewer.

13. This view is of course a `FramesocPart` and it is provided by and external plugin extending the `fr.inria.soctrace.framesoc.ui.perspective.part` extension point and advertising the view ID `FramesocViews.GANTT_CHART_VIEW_ID`.

raw trace events, in order to produce the states and links visualized in the Gantt. For these reasons, the FrameSoC Gantt Chart View, beside providing some default behavior for trace visualization, enables to highly customize trace representation. This is achieved through the modular architecture summarized in Figure 20, where FrameSoC modules are in light blue, the TChartsLite Gantt viewer is in orange and customizable modules are in green.

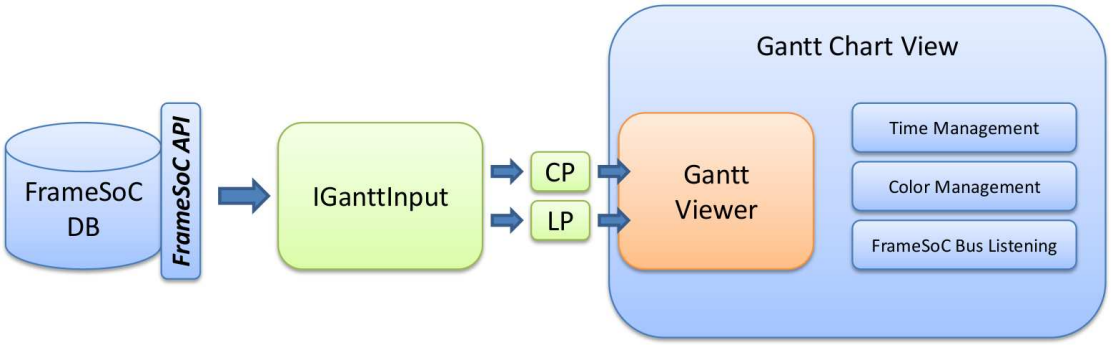


FIGURE 20 – Gantt Chart modular architecture

In this figure we observe that in the data flow going from data retrieval to visualization there are three different, decoupled, components: the raw data access modules, the input modules and the visualization modules.

Trace raw data are stored in the FrameSoC database according to the FrameSoC data-model [3, 5].

A customizable input module (implementing the `IGanttInput` interface) reads these data and adapt them for the TChartsLite viewer through a content provider (*CP*) and a label provider (*LP*), whose interfaces are defined by the TChartsLite viewer. This viewer, in fact, implements the standard JFace philosophy of decoupling how data is visualized from the data-model: basically it defines how it intends to request data (e.g., as a list, as a tree, etc.) through the content provider and how it intends to put them on the screen (e.g., as a string, as a figure, etc..) through the label provider [14]. Using the API defined in its content and label providers, the viewer simply visualize on the screen the data contained in the input module.

The `IGanttInput` interface also defines the methods used by the Gantt view to trigger trace loading and color update. In particular the method `loadTimeWindow()` prepares the Gantt model for a given time window and the method `updateAfterColorChange()` updates the model colors after a color change is notified on the FrameSoC Bus (see Section 3.2).

The FrameSoC Gantt plugin provides a default implementation of the `IGanttInput` and the TChartsLite providers. In this default implementation, the input module adapts the FrameSoC data-model to a *Gantt-friendly* model, which is a hierarchy of Gantt rows (`IGanttRow`), each of whom is composed by a list of events with duration, i.e., states (`IGanttEvent`). This adaptation process is done according to the following rules:

- Different `IGanttRow`s refer to different FrameSoC event producers.
- A FrameSoC punctual event becomes a `IGanttEvent` with null duration.
- A FrameSoC state becomes a `IGanttEvent` with non null duration.
- A FrameSoC link becomes a connection between two `IGanttEvent`s belonging to two different `IGanttRow`s.

14. For more information about JFace and its *model-view-controller* design, please refer to JFace documentation [12].

In order to provide a custom implementation of the `IGanttInput` and the TChartsLite providers, an external plugin has to provide an extension to the `fr.inria.soctrace.framesoc.ui.tclgantt.adapter` extension point, defined by the Gantt plugin. This extension point contains the following fields:

- `traceTypeName`
  The unique name of the trace type this adapter is related to. The trace type defines the trace format.
- `modelInput`
  Class implementing the `IGanttInput` interface.
- `contentProvider`
  Class implementing the TChartsLite content provider interface.
- `labelProvider`
  Class implementing the TChartsLite label provider interface.

If for a trace format an extension to this extension point is provided, the three corresponding classes are used. Otherwise the default implementation is used. Having the possibility to redefine the classes responsible of the adaptation process between the FrameSoC data-model and the Gantt visualization is useful. This is particularly clear, for example, for trace formats that are imported into FrameSoC using only punctual events: instead of having a state entity, they have two punctual events representing the start and the end of the state. In this case, the custom input class could rebuild the states starting from a punctual event representation, in order to have a semantically significant Gantt chart.

4 Conclusions

In this technical report we described the FrameSoC Workbench, a consistent graphical environment for trace management and analysis, built on top of FrameSoC infrastructure.

First, we provided a functional description of the workbench, presented from the user point of view. This workbench supports and facilitates trace analysis flow through a correlation among views and a global environment consistency, providing management and analysis views. Management views enable trace browsing and metadata viewing/editing. Analysis views provide for different visualizations of trace data: an event density chart, a statistics pie chart, a table of events, and a Gantt chart. These analysis views exhibit a common and consistent behavior regarding events like selection or highlighting, and are able to communicate among them.

Second, we provided a more technical description of the mechanisms supporting view correlation, environment consistency and modularity. The communication among views is achieved through a simple Publish-Subscribe mechanism, which also enables sharing variables. The consistent behavior of analysis views relies on a software design based on factorization of analysis view common functionalities. Modularity and easy extensibility of the workbench is achieved through the use of Eclipse extension point mechanism. A first extension point enables easy contribution of new analysis views. A second extension point allows for easy customization of Gantt chart visualization for specific trace formats.

Among our perspective there are, first of all, some improvements to existing analysis views, as for example the possibility to set event-type or event-producer filtering in both the event density chart and in the statistics pie chart, in order to compute the different metrics using only a subsets of events. Still considering the statistics pie chart, it would be useful to have the possibility to specify the time interval where statistics are computed. Regarding the Gantt chart, a missing feature is a zoom/dezoom able to perform controlled visual aggregation on graphical elements too large to be visualized with the given amount of pixels: in the current Gantt viewer, in fact,

when dezooming, several graphical elements (states, punctual events) may be collapsed in a single pixel, giving place to uncontrolled visual aggregation, where some information is lost (without the user being aware of this loss) while the graphical objects are still consuming memory. A better solution would be to control the aggregation in these situations, by using special graphical elements to identify aggregated zone, as done in Pajé [4] and in Lttng Eclipse viewer [6]: this way the user is aware of information loss and the memory for not displayed graphical elements can be saved. Then, we are thinking about adding new kinds of analysis views, as for example the visualization of metrics like the probability density function of state duration for one or more given type of states.

Finally, we always want to improve our analysis environment ergonomics, in order to provide a better user experience and an easier analysis. Going in this direction, we are interested in conceiving a diagnosis tool, which can report on the different user interactions with the FrameSoC workbench. Not only this would be beneficial from the ergonomics point of view, but the information could be used as a reporting tool about executed trace analyses. Such reports could be used for defining a default data analysis workflow and automating some sequences of actions.

References

[1] SoC-TRACE project. `http://tinyurl.com/minalogic-soc-trace`.

[2] Generoso Pagano, Damien Dosimont, Guillaume Huard, Vania Marangozova-Martin, and Jean-Marc Vincent. Trace Management and Analysis for Embedded Systems. In *Proceedings of the IEEE seventh International Symposium on Embedded Multicore SoCs (MCSoC-13)*, Tokyo, Japan, sep 2013.

[3] Generoso Pagano and Vania Marangonzova-Martin. SoC-Trace Infrastructure. Rapport Technique RT-0427, INRIA, November 2012.

[4] Jacques Chassin De Kergommeaux and Benhur De Oliveira Stein. Paje: An Extensible Environment for Visualizing Multi-Threaded Program Executions. In *Proc. Euro-Par 2000, Springer-Verlag, LNCS*, pages 133–144, 2000.

[5] Alexis Martin, Generoso Pagano, Jérôme Correnoz, and Vania Marangozova-Martin. Analyse de systèmes embarqués par structuration de traces d'exécution. In *ComPAS'2014*, Neuchâtel, Switzerland, April 2014.

[6] Lttng Eclipse Viewer. `http://lttng.org/eclipse`.

[7] J.M. Wilson. Gantt Charts: A Centenary Appreciation. *European Journal of Operational Research*, 149(2):430–437, 2003.

[8] STMicroelectronics. `http://www.st.com`.

[9] Generoso Pagano and Vania Marangozova-Martin. SoC-Trace Infrastructure Benchmark. Rapport Technique RT-0435, INRIA, June 2013.

[10] Sasu Tarkoma. *Publish/subscribe systems: design and principles.* Wiley series in communications networking & distributed systems. Wiley, Chichester, West Sussex, 2012.

[11] SWT: The Standard Widget Toolkit. `http://www.eclipse.org/swt/`.

[12] JFace. `http://wiki.eclipse.org/index.php/JFace`.

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-0803