EGSE architecture
and
the new generation of software technologies

M.Poletti [1], E. Schito.[1], E. Martinelli [1], D.Antonelli [2]

[1]Thales Alenia Space Italia
S.S. Padana Superiore, 290
20090 Vimodrone (MI) - Italy
mauro.poletti@thalesaleniaspace.com
elia.schito@thalesaleniaspace.com
egidio.martinelli@thalesaleniaspace.com

[2]Intecs SpA
Via Archimede, 10
20129 Milano - Italy
Dario.antonelli @intecs.it

INTRODUCTION

The aim of this paper is to explain the reason why we are thinking to a new EGSE architecture and the way this system can improve the efficiency in every phase of-board systems integration and validation.
As everybody know, a primary goal during satellite developing is to save time (and money) and, looking at the nominal workflow for satellite production, the large amount of test campaigns that are spread on the whole development cycle is one of the greatest time consuming activity. The main reason of this statement is due to the fact that every validation or integration process needs its own EGSE tools, different skills to conduct the test and a different approach to define the pass/fail criteria. In order to improve the global efficiency of the EGSE itself, we started the definition of new EGSE architecture that inherits most of the progress made in last years by software technology.

The Basic Architecture concept

This analysis starts from simple considerations: in the last 15 years the basic EGSE architecture has never changed: EGSE has taken advantages from the PC HW performances growth (Moore law), but the same cannot be said about software technologies. Focus has been made on functions to allow TM/TC processing to cope ECSS standards and with specific modifications to answer to dedicated needs of the EGSE operators in a suitable and effective manner. But the communication among the core modules are usually hard coded and cannot be easily changed. So, the common EGSE modules cannot be considered like generic services providers. They are highly configurable systems but not "open" systems.

The Basic MMI Concept

A simple metaphor could be used to better explain how a test engineer uses EGSE Test Conductor Console: the interaction between a computer user and the operating system is a "communication" exercise whose language (the Computer Desktop) is imposed by the manufacturers. The same applies in the EGSE environment: several core EGSEs are on the market and when they are made available to the test engineers, be sure that the most polite comment from them will be "This is working, but I would like to see also …".The fig. 1 clearly explain this concept.

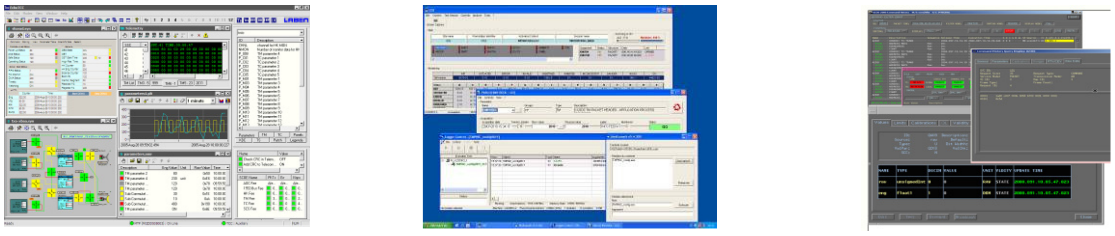


Fig. 1 TAS-I ECHO, TAS-F OCOE-5, ESA S2K MMIs

A NEW STARTING POINT

There is a cornerstone in every EGSE architecture which can be identified as “Common Core”. This component implements the fundamental functions that belongs to satellite EGSE and Control: Telemetry monitoring, Telecommand sending, Procedure Execution, Data Archiving, and so on. What is different within the various EGSEs are the way the results are presented, the Test Report are produced and the different tools used for Data Retrieval and Post Processing Analysis.

Within this context if the EGSE User is satisfied by the Common Core functionality, he still have many wishes about the way to “communicate” with the EGSE.

The WWW as Model

We will use the World Wide Web as a wide known model to rethink how to describe EGSE component. The WEB is, from the user point of view, the software domain that manifest the highest growth in terms of functionality, contents and new technologies. So, taking into consideration how the information is produced and consumed in Internet we have analyzed the Client/Server model of the WEB versus the Test Conductor Console/Common Core model of the EGSE.

The WEB Client Server Model

The client server model of the WEB can be categorized into three parts:

- User Interface
- Business or Application Logic
- Data Management

Traditional application

Traditional web application development has distributed the implementation of the user interface across the network, with much of the user interface logic and code executed on the server (thin client, fat server).

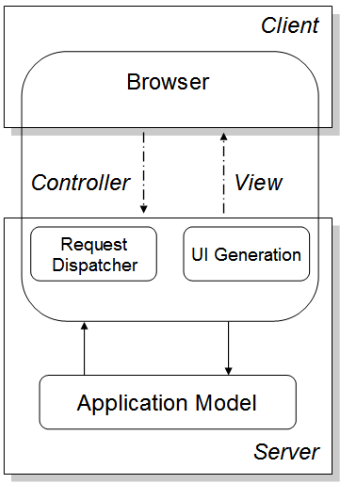


Fig. 2 Traditional WEB application

This has several key problems:

- Poor distribution of processing – With a large number of clients, doing all the processing on the server is inefficient.
- Difficult programming model – Programming a user interface across client/server is difficult. When every interaction with a user must involves a request/response, user interface design with this model is complicated and error prone.

Service Oriented Application (UI Separation)

The evolution, primarily due to the Ajax movement, has been related to moving the user interface code to the client. The new browser platforms and the availability of HTTP client capabilities, have allowed much more comprehensive client side user interface implementations. However, with these new found capabilities, it is important to understand how to build client/server applications.

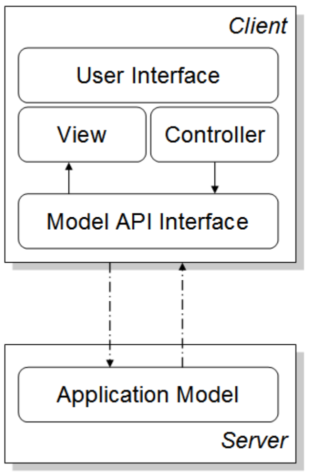


Fig. 3 Service Oriented Application

So how do we decide what code should run on the client and what should run on the server? Quite simply, user interface code is best run on the client side, while application/business logic and data management is best run on the server side.

Taking a valuable lesson from Object Oriented programming to guide this model we create objects that encapsulate most of their behavior and have a minimal interface area. Designing for a modular reusable remote interface is one of the primarily goal of a Service Oriented Architecture (SOA); data are communicated with a defined API, rather than incoherent chunks of user interface.

The advantages of a clean client/server model where user interface logic and code is delegated to the client (browser) can be summarized as:

- Scalability – It is quite easy to observe the significant scalability advantage of client side processing. The more clients use an application, the more client machines that are available, whereas the server processing capabilities remain constant (until you buy more servers).
- Organized programming model – The user interface is properly segmented from application business logic.
- Client side state management – Maintaining transient session state information on the client reduces the memory load on the server. This also allows clients to leverage more RESTful interaction which can further improve scalability and caching opportunities.
- Offline applications – If much of the code for an application is already built to run on the client, the creation of an offline version of the application will certainly be easier.
- Interoperability – By using structured data with minimal APIs for interaction, it is much easier to connect additional consumers and producers to interact with existing systems.

Passing from model to architecture, the software paradigm that facilitate this client/server implementation is the **Model View Controller (MVC);** an architectural pattern widely used in graphic interface development.

THE MODEL VIEW CONTROLLER

The basis of the MVC are the following:

- The Model represents the application data
- The View renders a presentation of model data

- The Controller handles and routes requests made by the client

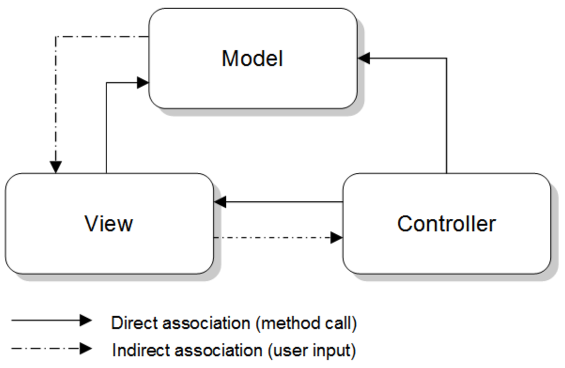

Model
View
Controller
Direct association (method call)
Indirect association (user input)

Fig. 4 Model View Controller Iteration diagram

The Model

The model is used to manage the information and notify observers when that information changes. The model is the domain-specific representation of the data upon which the application operates. Domain logic adds meaning to raw data (for example, calculating parameters). When a model changes its state, it notifies its associated views so they can be refreshed.

Many applications, including EGSEs, use a persistent storage mechanism such as a database to store data. MVC does not specifically mention the data access layer because it is understood to be underneath or encapsulated by the model.

The View

The view renders the model into a form suitable for interaction, typically a user interface element. Multiple views can exist for a single model for different purposes. A viewport typically has a one to one correspondence with a display surface and knows how to render to it.

The Controller

The controller receives input and starts a response by making calls on model objects. A controller accepts input from the user and instructs the model and viewport to perform actions based on that input.

An MVC application may be a collection of model/view/controller triplets, each of them is responsible for a different UI element.

Summarizing, the separation between Model, View and Controller can lead to the development of the different functions by different subjects. If the Model export data through standard API (REST, SOAP, JSON) allows external developers to implement different and flexile views. Who is in charge of developing the View-Controller, interrogates the server, load raw data and arrange (or compute) them in innovative way. If more subjects develop different views using the same data, the final user can choose the more suitable, usable and convenient interface within the reference context.

How SOA and MVC fits with EGSE

Now, the big question is: how all these theory fits with EGSE? If we represent an EGSE using the previous Client/Server Model we have the Common Core (the Server) connected to Test Conductor Console (the Clients) and, using the topics previously explained, we can easily state that if we should have a Common Core based on Service Oriented Architecture (SOA) we could easily use an MVC pattern to obtain:

- Interoperability – By using structured objects with minimal APIs for interaction, it is much easier to connect different modules with dedicated data processing. Moreover this architecture will allow a “Module Competition” business schema perfect for SME.

- Offline applications and reporting – If much of the code for an application is already built to run on the client, it is easier to create different offline view of the data as well as dedicated reports.
- Scalability – It is quite easy to observe the significant scalability advantage of client side processing. As soon as that the Satellite integration evolves from equipment to subsystem and system level, the more clients will use an application, the more client machines will be available, whereas the server processing objects will grow up accordingly.

The New Architecture Proof of Concept

An excellent test for a good client/server model is how easy is it to create a new user interface for the same application so we tried to prove this statement concentrating our effort on the View part of the MVC pattern and leaving unchanged the communication API, even if the EGSE Common Core is not (yet) based on SOA. The next figure explain the model Test Conductor Console/Common Core used in our exercise.

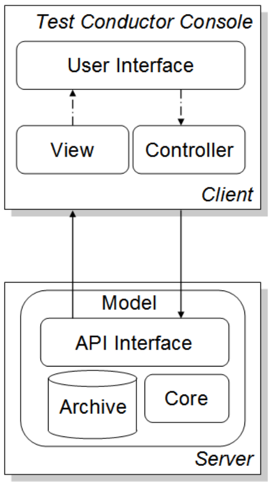


Fig. 5 EGSE Client/Server Model

Since most of the innovative part of the technology is relevant to the WEB, we have done this programme in cooperation with the Information Technology department - Milano University. The goal is to obtain a Proof of Concept of a rapid prototyping to Man Machine Interface generation, taking the maximum profit of HTLM5 standards, dynamic languages (Ruby), and web frameworks (Ruby on Rails ®) and JavaScript library (jQuery).

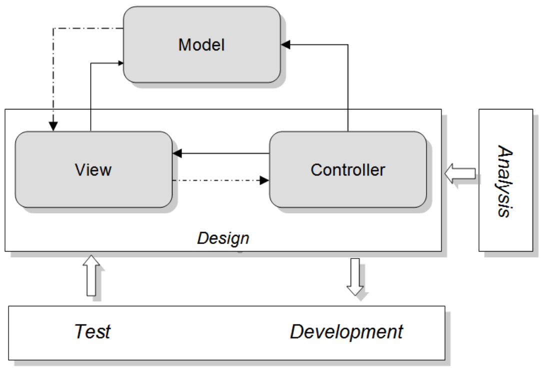
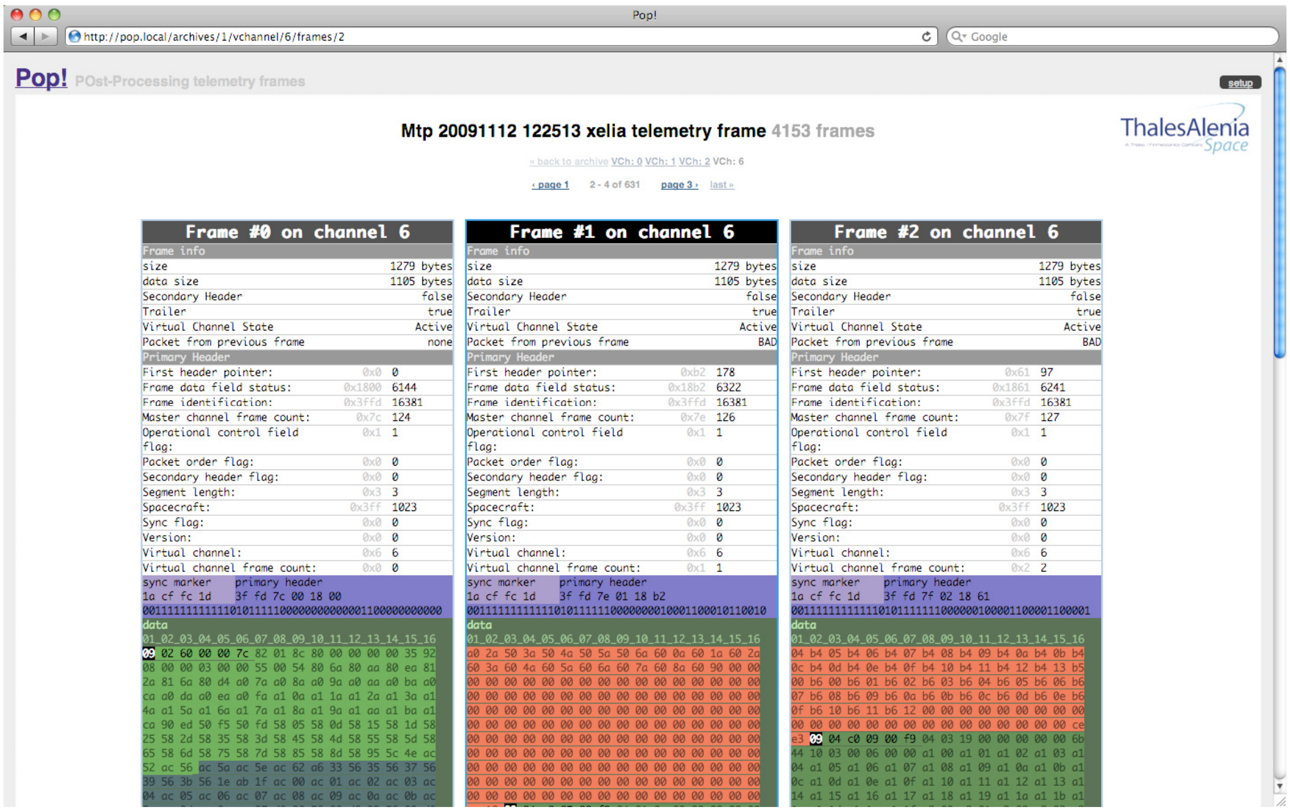


Fig. 6 Development process based on user iteration.

The basis is to implement a highly interactive process between the developer and the user in order to jointly define the MMI, and rapidly generate it using the technology defined above. Two example (limited by the actual development status) are showed below: both are off-line data presentation. The first one is a TLM frame analyzer and a Packet TLM decommutator, the second is a parameter trend analysis display. Both programs use the EGSE archive files as data source. The examples are showed in Fig 7 and 8.



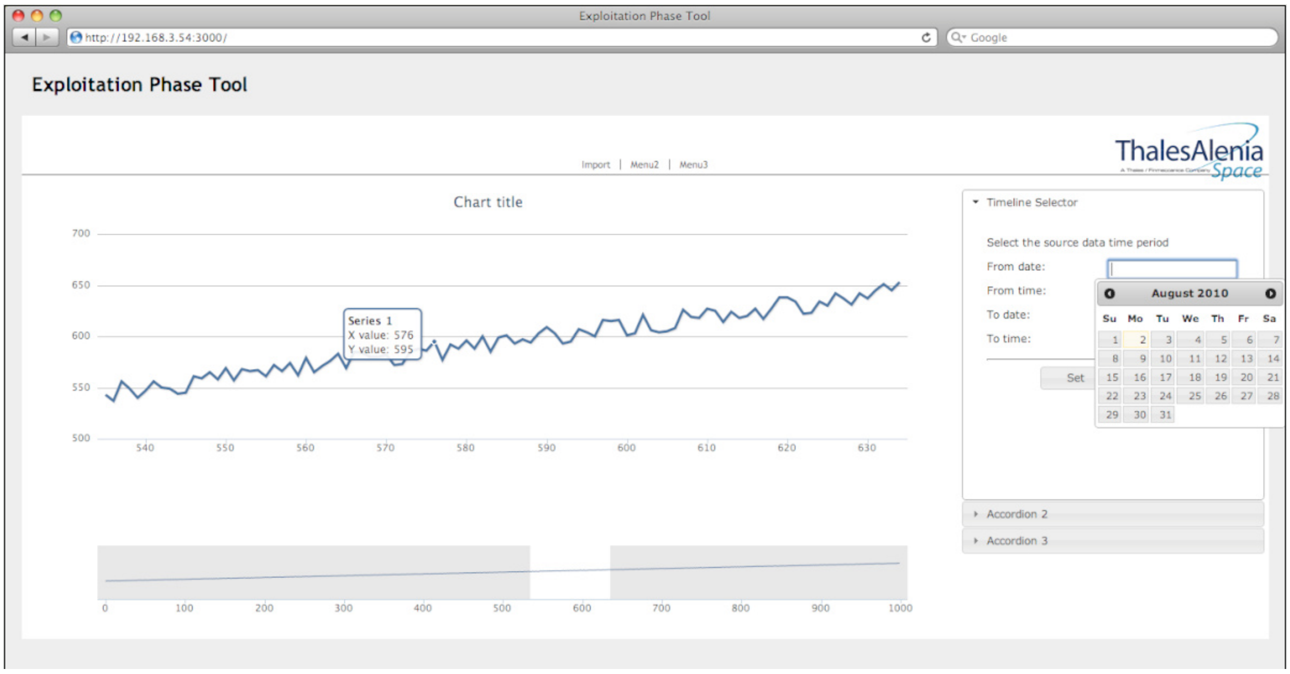
```
Pop!
http://pop.local/archives/1/vchannel/6/frames/2

Pop! POst-Processing telemetry frames                                   setup

Mtp 20091112 122513 xelia telemetry frame 4153 frames
« back to archive VCh: 0 VCh: 1 VCh: 2 VCh: 6
‹ page 1    2 - 4 of 631    page 3 ›  last »
```

```
Frame #0 on channel 6
Frame info
size                                    1279 bytes
data size                               1105 bytes
Secondary Header                             false
Trailer                                       true
Virtual Channel State                       Active
Packet from previous frame                    none
Primary Header
First header pointer:              0x0      0
Frame data field status:           0x1800   6144
Frame identification:              0x3ffd   16381
Master channel frame count:        0x7c     124
Operational control field flag:    0x1      1
Packet order flag:                 0x0      0
Secondary header flag:             0x0      0
Segment length:                    0x3      3
Spacecraft:                        0x3ff    1023
Sync flag:                         0x0      0
Version:                           0x0      0
Virtual channel:                   0x6      6
Virtual channel frame count:       0x0      0
sync marker     primary header
1a cf fc 1d     3f fd 7c 00 18 00
001111111111110101111100000000000000001100000000000
data
01_02_03_04_05_06_07_08_09_10_11_12_13_14_15_16
09 02 60 00 00 7c 82 01 8c 80 00 00 00 00 35 92
08 00 00 03 00 00 55 00 54 80 6a 80 aa 80 ea 81
2a 81 6a 80 d4 a0 7a a0 8a a0 9a a0 aa a0 ba a0
ca a0 da a0 ea a0 fa a1 0a a1 1a a1 2a a1 3a a1
4a a1 5a a1 6a a1 7a a1 8a a1 9a a1 aa a1 ba a1
ca 90 ed 50 f5 50 fd 58 05 58 0d 58 15 58 1d 58
25 58 2d 58 35 58 3d 58 45 58 4d 58 55 58 5d 58
65 58 6d 58 75 58 7d 58 85 58 8d 58 95 5c 4e ac
52 ac 56 ac 5a ac 5e ac 62 a6 33 56 35 56 37 56
39 56 3b 56 1e ab 1f ac 00 ac 01 ac 02 ac 03 ac
04 ac 05 ac 06 ac 07 ac 08 ac 09 ac 0a ac 0b ac
```

```
Frame #1 on channel 6
Frame info
size                                    1279 bytes
data size                               1105 bytes
Secondary Header                             false
Trailer                                       true
Virtual Channel State                       Active
Packet from previous frame                     BAD
Primary Header
First header pointer:              0xb2     178
Frame data field status:           0x18b2   6322
Frame identification:              0x3ffd   16381
Master channel frame count:        0x7e     126
Operational control field flag:    0x1      1
Packet order flag:                 0x0      0
Secondary header flag:             0x0      0
Segment length:                    0x3      3
Spacecraft:                        0x3ff    1023
Sync flag:                         0x0      0
Version:                           0x0      0
Virtual channel:                   0x6      6
Virtual channel frame count:       0x1      1
sync marker     primary header
1a cf fc 1d     3f fd 7e 01 18 b2
001111111111110101111110000000010001100010110010
data
01_02_03_04_05_06_07_08_09_10_11_12_13_14_15_16
a0 2a 50 3a 50 4a 50 5a 50 6a 60 0a 60 1a 60 2a
60 3a 60 4a 60 5a 60 6a 60 7a 60 8a 60 90 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
Frame #2 on channel 6
Frame info
size                                    1279 bytes
data size                               1105 bytes
Secondary Header                             false
Trailer                                       true
Virtual Channel State                       Active
Packet from previous frame                     BAD
Primary Header
First header pointer:              0x61     97
Frame data field status:           0x1861   6241
Frame identification:              0x3ffd   16381
Master channel frame count:        0x7f     127
Operational control field flag:    0x1      1
Packet order flag:                 0x0      0
Secondary header flag:             0x0      0
Segment length:                    0x3      3
Spacecraft:                        0x3ff    1023
Sync flag:                         0x0      0
Version:                           0x0      0
Virtual channel:                   0x6      6
Virtual channel frame count:       0x2      2
sync marker     primary header
1a cf fc 1d     3f fd 7f 02 18 61
001111111111110101111111000000100001100001100001
data
01_02_03_04_05_06_07_08_09_10_11_12_13_14_15_16
04 b4 05 b4 06 b4 07 b4 08 b4 09 b4 0a b4 0b b4
0c b4 0d b4 0e b4 0f b4 10 b4 11 b4 12 b4 13 b5
00 b6 00 b6 01 b6 02 b6 03 b6 04 b6 05 b6 06 b6
07 b6 08 b6 09 b6 0a b6 0b b6 0c b6 0d b6 0e b6
0f b6 10 b6 11 b6 12 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ce
e3 09 04 c0 09 00 f9 04 03 19 00 00 00 00 00 6b
44 10 03 00 06 00 00 a1 00 a1 01 a1 02 a1 03 a1
04 a1 05 a1 06 a1 07 a1 08 a1 09 a1 0a a1 0b a1
0c a1 0d a1 0e a1 0f a1 10 a1 11 a1 12 a1 13 a1
14 a1 15 a1 16 a1 17 a1 18 a1 19 a1 1a a1 1b a1
```

Fig. 7 Telemetry Frame analysis and Packet decommutation



```
Exploitation Phase Tool
http://192.168.3.54:3000/

Exploitation Phase Tool
Import | Menu2 | Menu3

Timeline Selector
Select the source data time period
From date:
From time:
To date:
To time:
Set
Accordion 2
Accordion 3
```

Fig. 8 Parameter Plot (trend analysis and detailed view)

Actually the computer browser is used for the data presentation, but we are also investigation the possibility to create final desktop application using dedicated Ruby framework like Bowline.

Conclusions

We have showed, through the “Prof of Concept”, that software architectures used in totally different context can be successfully used within EGSE to improve user experience. This increases our belief that Model View Controller, together with a Service Oriented Architecture will constitute the basis for really new EGSE generation

Reference

Without *Wikipedia © The Free Encyclopedia* this article never would have been written