PROTOTYPING FOR NEW CCS TECHNOLOGIES

Claude CAZENAVE

Astrium Satellites
31 Rue des Cosmonautes
31402 TOULOUSE
claude.cazenave@astrium.eads.net

ABSTRACT

Astrium Satellites is highly contributing to the development of the next generation CCS project, currently named EGS-CC, with the objective of a European harmonisation applicable to all primes, agencies in the scope of EGSE and Control Center.

In the past, ASTRIUM Satellite has developed 2 different CCS product lines to cope with different needs:

- a very generic product compatible with EGSE and control centre needs : Open Center
- a light weight approach compatible with lightweight EGSEs like Software Verification Facilities : SimTG SimOPS

To propose its best contribution to EGS-CC, Astrium Satellites is currently prototyping technologies to be proposed in the frame of EGS-CC. For the best analysis, Astrium Satellites prototyping is performed in a first step at technology level and then are integrated into our lightweight solution, SimTG SimOPS [1], for early user validation.

The recommended technologies are in particular around the topics of middleware, test language and closer integration with the system database. The current developments are thus focusing on:

- ZeroMQ : a middleware technology, message oriented, used to distribute TM parameters to the various applications in a very efficient way. This technology enforces parallel programming and provides the smallest communication overhead depending on the need (collocation in the same process, in the same machine or no collocation at all).
- Java: a well-known software language to be used as test language. This language benefits of the state of art development environment (editors, debugger). This brings at no cost a solution compatible with the requirements and very mature.
- EMF: a generative approach to integrate the EGS-CC M&C data model. By using the same data model, we can fully generate the run time data model for EGS-CC without needing to develop intermediate data structures and the related SW. It simplifies the import of data from the system data base (e.g. RangeDB for ASTRIUM Satellite). In addition, combined with the previous technology, we can generate a compiled TM/TC interface enabling on the fly checking (i.e. during the edition and without compiling) of TM/TC use in the test sequences.

This paper will provide the technical approach / selected technologies and the results / recommendations learnt during these prototyping phases.

INTRODUCTION

From our long term experience in developing CCS products, Open Center or SimTG SimOPS, Astrium Satellites has a good knowledge of the cost to maintain these software applications. This maintenance cost is not only at developer level but also at user level due to the complexity of these systems and the need to provide immediate fixes in critical phases of spacecraft development. In particular, we consider that the configuration of a CCS for a given usage is a key issue in the maintenance cost. This brings a first recommendation for technology and architecture choices: keep the configuration simple as much as possible and for complex things do not reinvent the wheel and rely on well proven technologies for what concerns management of complexity. Concerning configuration, the link with other sources of information like the system database have been a huge source of cost and delay. This has to be tackled by thinking about technologies that are either common or provide efficient bridges to exchange information.

In addition, this experience shows that the real time performance of these systems is always a main concern. This applies not only to the running phase but also to the setup/initialization phases or to the analysis phase. This brings a second recommendation to use, when possible, the fastest technology for these critical parts. The middleware used to exchange information between the various EGSE and Control Center computers is clearly the most critical part.

Last but not least the use of state of art technologies, already well adopted by the computer users, is an approach leading to a better adoption of the tools we develop. This only concerns the parts visible to the end user like graphical man machine interfaces and test procedures. Java language, since many years, has been selected for SimTG SimOPS test environment, as an alternative to specific languages part of the current CCS. This language is well known, often taught and takes advantages of state of the art tools and technologies for editing and debugging. Thus it is also a recommendation to use it for a new CCS but with a revised approach to take into account user's feedback and to take benefits of a better integration with the database.

As a result of these recommendations, Astrium Satellites is prototyping the following technologies:

- Zero MQ for data distribution
- EMF for interoperability with new system databases
- Java for test procedures with a revised approach

The next chapters will introduce each technology and the results of the related prototyping phase.

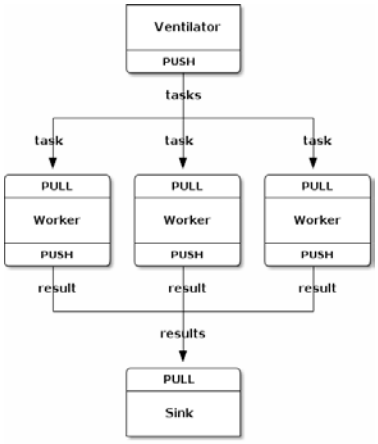
ZEROMQ

Extract from Wikipedia : ØMQ

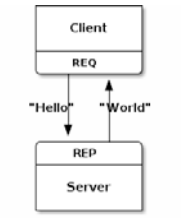
ØMQ (also spelled ZeroMQ, 0MQ or ZMQ) is a high performance asynchronous messaging library aimed at used in scalable distributed or concurrent applications. It provides a message queue, but unlike message oriented middleware, a ØMQ system can run without a dedicated message broker. The library is designed to have a familiar socket-style API.

This well summarizes the reasons why we selected this technology [2].

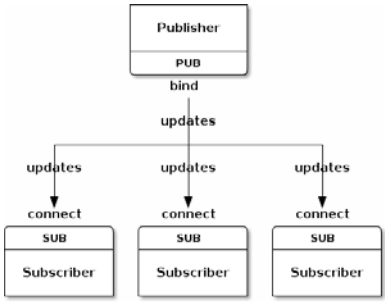
First of all, it is high performance and indeed we measured that it is faster than other messaging technologies based on JMS Standard (rabbitMQ, ActiveMQ, ...). The prototype demonstrated we can distribute more than 10Mbytes of raw data per second to several computers (on Astrium Satellites LAN, i.e. including routers) while keeping a very low CPU and memory consumption on the computer providing these data. One of the reasons why this technology is faster is that it relies on very low level (it is also why it has a socket style API). This has some drawbacks since it does not provide out of the box marshalling of the data. But it gives the freedom to perform our own wrapping with a tuning very specific to our use cases. It is important to note that in all cases a wrapping of such technology is mandatory since they are subject to changes. Compared to the previous designs where the data exchange was performed using CORBA middleware, this message oriented technology provides more flexibility to perform parallelized computation and data distribution without having to take care about multi-threading issues. In addition, an optimized protocol can be used according to the needs: inter thread exchange, inter process exchange on the same computer or inter computer exchange. The next figure describes some typical use cases of message oriented architectures.



Parallel pipeline with PUSH/PULL



Request and Reply



Publish-Subscribe

Fig1: Use of message oriented architectures

The second reason for selecting this technology is that it can be used without a message broker. This is very useful when we want to configure a CCS system on a single computer and if possible in a single process. Use cases like Software Verification Facility will take benefit of such simple architecture for both the setup time and the acceleration factor on batch mode test cases (for instance when connected to a numerical simulator).

The architecture we defined for the prototyping was first targeted to the use case of a lightweight CCS connected to a numerical simulator. A lightweight CCS is a software application (i.e. one process) which provides a man machine interface but can also run in batch mode (i.e. without graphical interface). The architecture was then scaled to a distributed use case described in the next figure.

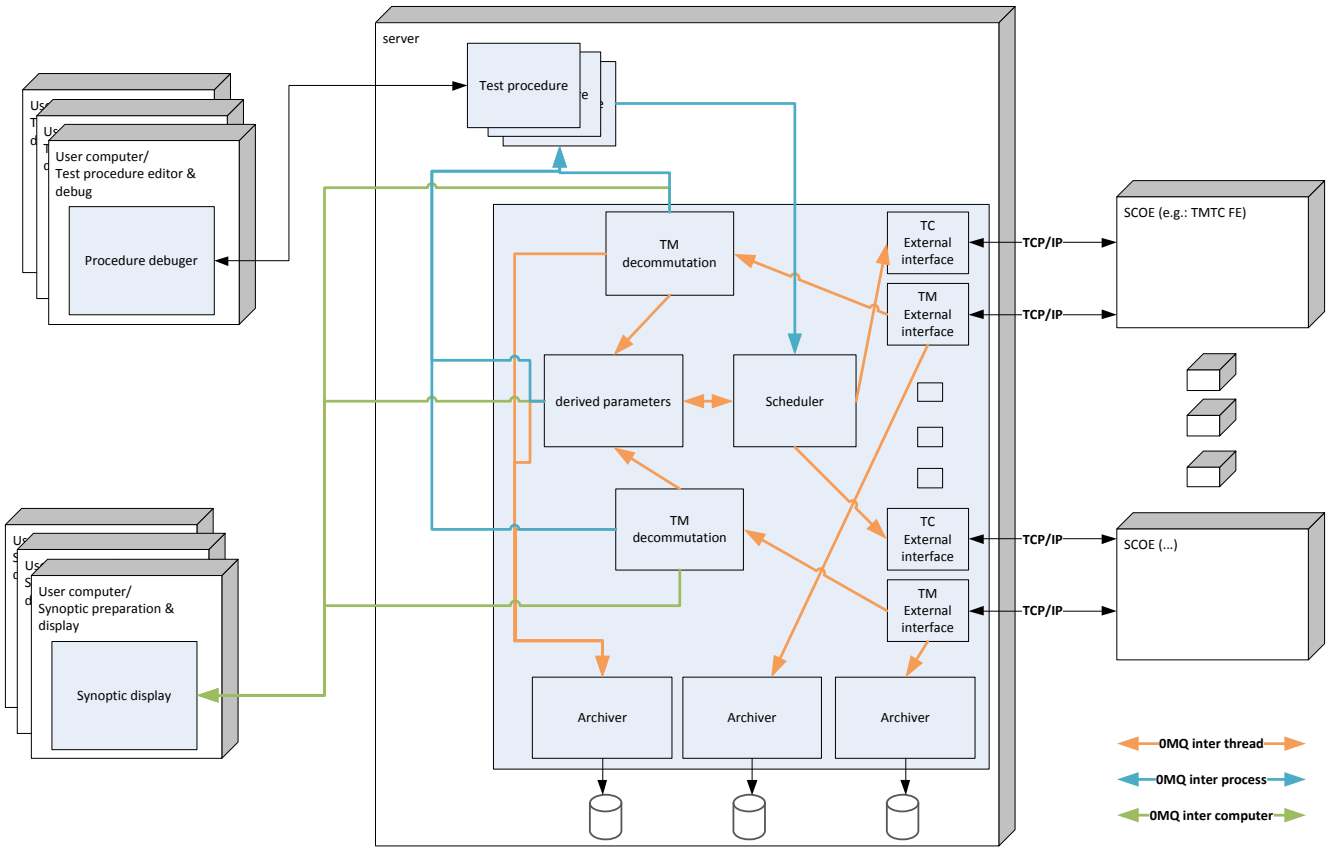


Fig2: Targeted architecture

Without changing the synoptic (user defined displays) already in use at Astrium Satellites, we integrated data flows coming from simulated SCOEs to analyse the behaviour of the system using both local and distributed ZeroMQ communication links. This prototype demonstrates that both the performance and the configuration aspects are compliant to the needs.

EMF

EMF stands for Eclipse Modelling Framework [3]. It provides its own modelling language, eCore, and automatic generation of Java source code of a model defined in that language. Astrium Satellites is using this technology to implement its new System Reference Data Base: RangeDB. Here is a quick summary of the Model Based approach used to develop RangeDB.

The starting point is the UML model, also called meta-model, of the data to be managed. For instance this model defines that a TC packet relies on a generic packet definition which contains a header, a data field and a trailer and that all these packet elements are defined using container of parameters. The next diagram is a simplified UML model for TM/TC packet and parameters.

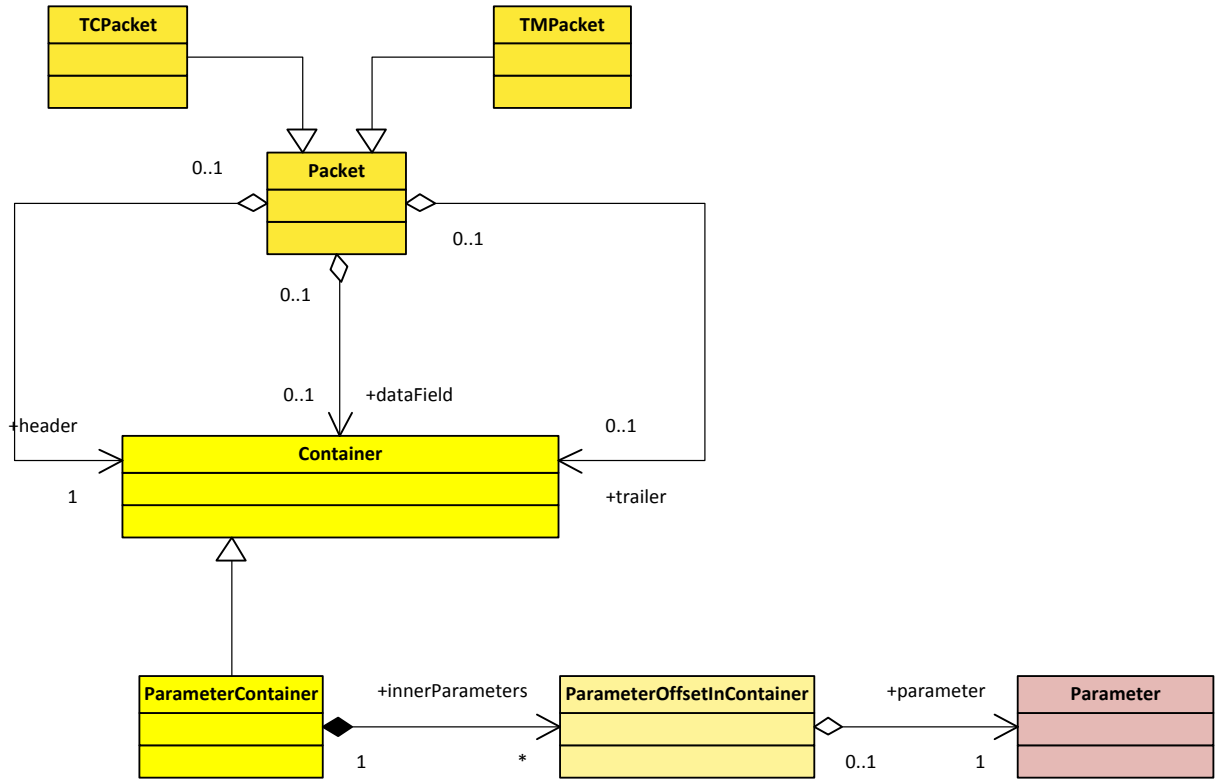


Fig3: Simplified UML model for TM/TC packet and parameters

EMF, thanks to a lot of customization features, allows to automatically generate all the Java classes used to persist the definition of these TC packets and the man machine interface to edit this definition. Here is an example of RangeDB user interface automatically generated thanks to EMF.

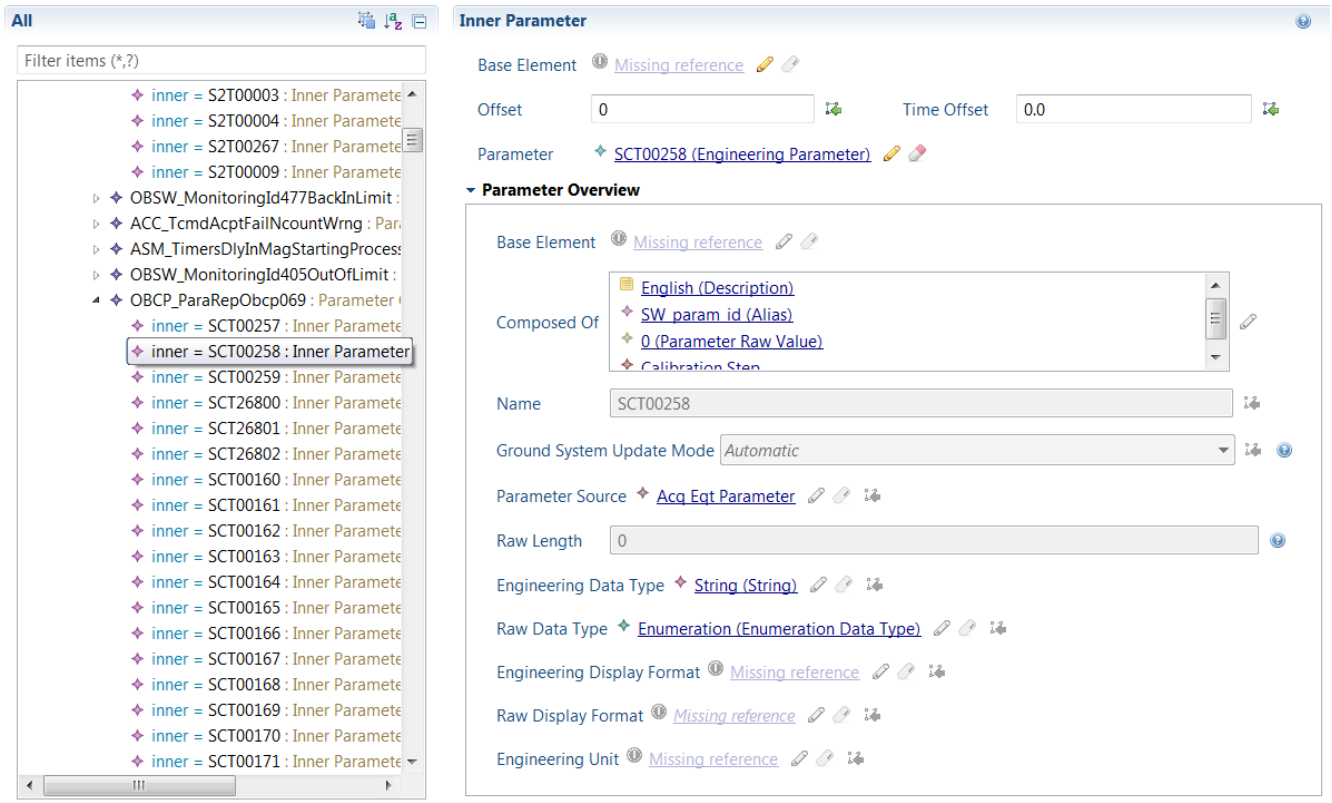


Fig4: RangeDB packet/parameter editor

In this context, it is worth to mention two additional technologies used for RangeDB together with EMF:

- OCL [4]: the Object Constraint Language is a language to define constraints to be verified by data stored in structures defined with a meta-model (e.g. UML or eCore). Once defined these constraints are automatically embedded in the application and provide reports and error highlighting in the editors with description of the not verified constraint.
- QVTO [5]: a language dedicated to model transformation. It operates on EMF/eCore models and allows transforming data defined in one EMF model into data defined in another EMF model. It provides imperative operations to structure the transformation and relies on OCL for model navigation. This is used to import or export data to/from RangeDB like for instance SCOS2000 MIB data.

Based on the experience acquired within RangeDB, the same approach, with a different customisation of the code generation, is applicable to EGS-CC to generate runtime software for sending TCs and for TM decoding. In addition, since we are using a RangeDB model compliant with the one defined in EGS-CC System Engineering Team Conceptual Data Model [6], the transformation of data defined in RangeDB into data compatible with EGS-CC should be rather straightforward.

The next chapter describes the use of Java procedures sending TCs according to an interface generated from a system database.

JAVA

Java is already used for writing test procedures in SimTG SimOPS tool [1]. This has given full satisfaction to the end users because it is a well-known language, it is object oriented (and this is quite helpful for software validation) but easy to learn and it ptovides many solution for integration with other software elements (database, scripts,…). Using the current implementation,TC sending with Java procedures, is similar to figure 5:

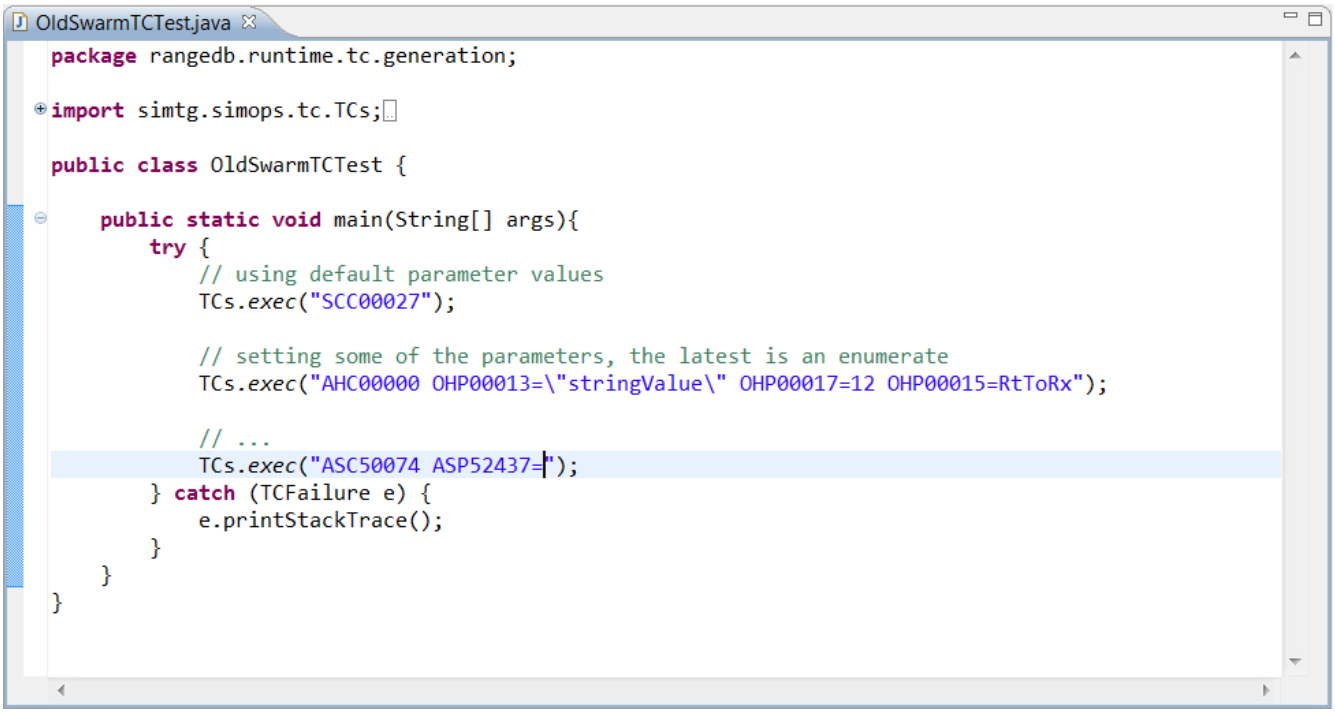


Fig5: *Sending TCs with current version of SimTG SimOPS*

The limitation of the test sequence example here above is that it does not check the validity of the TC name nor its parameters until the TC is sent. In other words the procedure is not checked at compilation time but at execution time. It is not a huge problem during the writing of the procedure since the tool provides TC browser that allows users to drag and drop TC names and TC parameter names. The problem is more when TC definition changes because it is not easy to check where the invalid names are. In addition there is no check on the parameter types (integers, real, strings or enumerates) and this is error prone.

So, to improve the current situation, the idea is to provide a so called “executable interface” of the TCs defined in the data base. An “executable interface” is a compiled interface that can be used to check, at compilation time, the validity of its use. Nowadays, state of art editors for Java language provides on the fly compilation of the source code. This brings an immediate feedback in terms of validity of the procedure to the writer. Moreover it provides a contextual help to discover what the TC parameters are and what the possible choices for enumerated values are. The next figure illustrates the writing of the same TCs using the new approach.

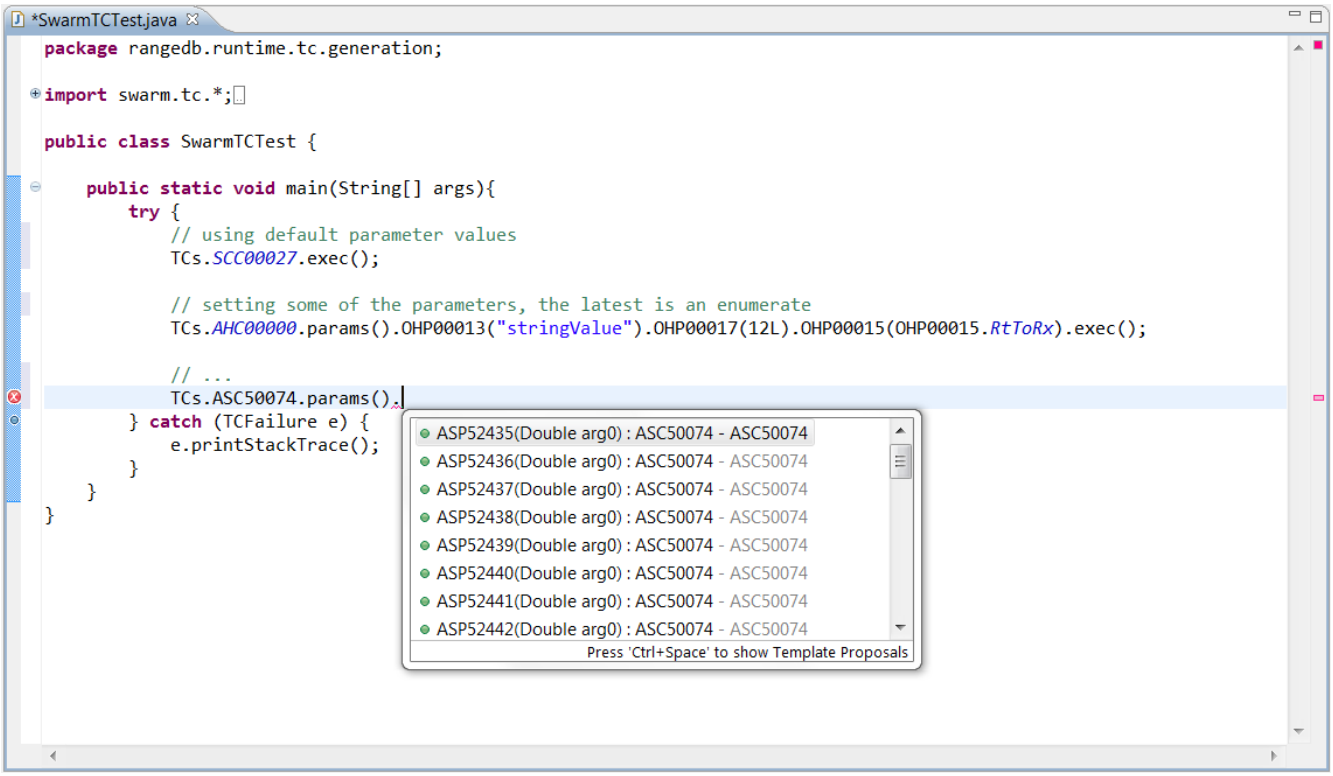


Fig6: *Sending TCs with future version of SimTG SimOPS*

An executable interface is a set of Java classes that are generated automatically from the data base. We have prototyped this code generation using a byte code generator to ensure that both the compilation time and the memory foot print of the TCs are the lowest. ASM [7] is the Java byte code generator used for the prototype. It was selected because it provides a powerful Eclipse plugin to ease the writing of the Java source code in charge of the generation of the Java byte code. As a result, it takes less than 10 seconds to generate 3000 TC Java classes from their definition in RangeDB tool (including the loading of the database) and the memory foot print average for each TC (it depends on the number of parameters) is 3Kbytes. It means a total around 9Mbytes for the TCs, without any compression, and it is considered as small enough to ensure the scalability of the system.

CONCLUSION

The results achieved in this prototyping phase are fully in line with the expectations. Astrium Satellite will continue the integration of these technologies in our lightweight CCS environment and will provide analysis of other technologies that have to be selected for EGS-CC. Here is the list of the domains to be analysed sorted according to their priority (highest first):

- Component Framework
- Data archiving
- User Defined Displays
- Scripting language
- Service integration platforms

REFERENCES

[1] SimTG SimOPS User Manual (SIMTG-UM-0010-ASTR), Issue 1.0, 23-Jan-2012

[2] ZeroMQ : http://www.zeromg.org

[3] EMF: http://www.eclipse.org/emf

[4] OCL: http://www.omg.org/spec/OCL/2.0

[5] QVTO: http://www.eclipse.org/m2m

[6] EGS-CC System Engineering Team Conceptual Data Model (EGSCC-SET-TN-1004), Issue 1.1, 30-05-2012

[7] ASM: http://asm.ow2.org