UMF – A Productive SMP2 Modelling and Development Tool Chain

Ellsiepen, P.[(1)], Fritzen, P.[(1)], Reggestad, V.[(2)], Walsh, A.[(2)]

[(1)]Telespazio VEGA Deutschland GmbH
Europaplatz 5, 64293 Darmstadt, Germany
E-mail: peter.ellsiepen@vega.de, peter.fritzen@vega.de

[(2)]European Space Operations Centre
Robert-Bosch-Str. 5, 64293 Darmstadt, Germany
E-mail: vemund.reggestad@esa.int, anthony.walsh@esa.int

ABSTRACT

Following the successful application of the SMP2, REFA and EGOS-MF technologies in various operational spacecraft simulators, ESOC have decided to consolidate their existing products into a Simulator Software Development Environment (SimSDE), thereby focussing on integration and usability aspects in order to further boost development productivity.

The main outcome of this activity is version 2 of the Universal Modelling Framework (UMF) product, which is planned for release this year. UMF v2 brings together the modelling features of EGOS-MF with the development features of UMF v1/SIMSAT MIE, putting the aspects of integration and usability at the centre. Simulation models are typically designed using a fully integrated UML tool, hiding the complexities of SMP2 via a model based approach. Alternatively, existing artefacts such as SMP2 Catalogues, Packages and Schedules from other SMP2 tools may be seamlessly imported into the UMF environment. As a result, UMF provides a productive end-to-end modelling and development tool chain that teams can use for creating large scale simulations such as an operational spacecraft simulator. UMF runs on both Linux and Windows and is independent of the simulation run-time environment, i.e. it can in principle be used together with existing SMP2 compliant run-time environments such as e.g. SIMSAT, BASILES, or EuroSim.

In this paper we describe the main concepts and capabilities of the UMF v2 tool chain in further detail. As a case study we present its usage in the development of the Bepi Colombo Simulator (BCSIM), a large scale operational spacecraft simulator that builds on UMF v2 as its baseline development environment together with the REFA v2 and GENM v5 products from SimSDE.

INTRODUCTION

Simulator Software Development Environment

The Simulator Software Development Environment (SimSDE) combines a number of ESOC products to form a coherent and consistent development environment for simulator developments:

- Universal Modelling Framework (UMF), version 2
- Library of Models (LoM), version 1
- Spacecraft Simulator Reference Architecture Specification (REFA), version 2
- Generic Models (GENM), version 5
- MOIS to JavaScript Converter (M2J), version 1

The main ingredient is the Universal Modelling Framework (UMF) which provides all concepts, mechanisms and tools for SMP2 based simulation modelling and development. UMF is complemented by the LoM, a new facility to foster exchange of SMP2 models via a model repository. For use in ESOC operational spacecraft simulators, REFA defines standard architectural patterns and interfaces while GENM contributes a number of generic model implementations and test tools. Finally, M2J provides a converter to enable reuse of MOIS operational procedure definitions in simulator test scripts for SIMSAT.

It should be noted, however, that the use of LoM, REFA, GENM, M2J and SIMSAT is optional, so UMF can also be used as stand-alone tool chain for arbitrary SMP2 based simulation developments.

Universal Modelling Framework

UMF v2 provides an Eclipse [5] based Integrated Development Environment (IDE) for SMP2 [1] based simulator developments, taking together "the best of all worlds" in combining features and tools from EGOS-MF [2], UMF v1 [3] and the E40-07 SMP Conformance Suite (SMP-CS) [4].

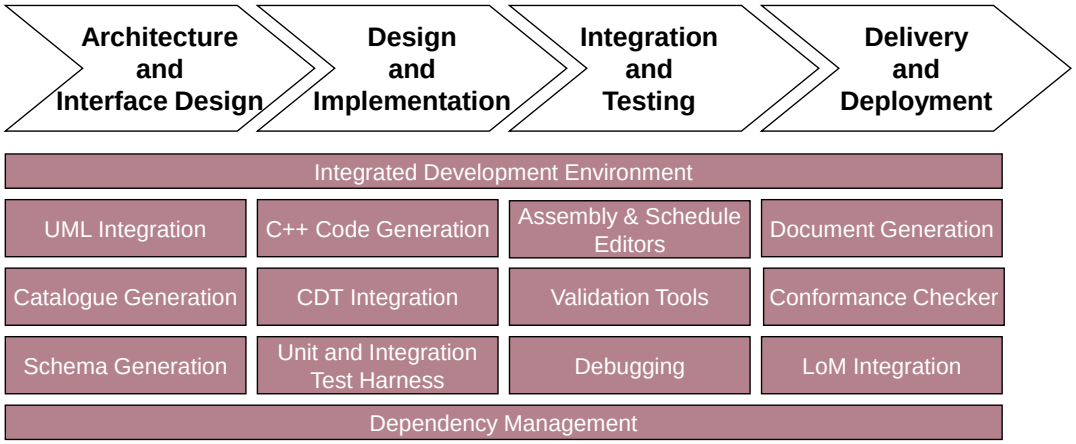

Fig. 1. UMF v2 combining EGOS-MF, UMF v1 and SMP-CS into a single IDE*: UMF mechanisms and tools support the full lifecycle of SMP2 based simulation developments.*

All UMF tools are available via both a Graphical User Interface (GUI) and on the command-line. The GUI enables fully integrated SMP2 development while the command-line tools allow, for example, batch processing or the automation within a modern Software Engineering approach such as continuous integration with automated quality analyses and automated testing.

A new Dependency Management facility spans the whole simulation lifecycle (from design to deployment) and ensures that all artefacts belonging to a package of models can be deployed as separate entities with clearly defined external dependencies (modularisation) – a prerequisite for model exchange via the new LoM facility.

UMF CONCEPTS AND FEATURES

The main ingredients of UMF are a) an integrated UML modelling tool with SMP2 customizations to enable seamless simulation modelling, b) a set of importers to bootstrap the design process in UML (e.g. requirements import) or to allow reuse of existing SMP2 artefacts (e.g. catalogue import), c) a set of validators that help identify problems early and allow navigating to the source of the problem with a single click, d) a set of generators, including documentation, catalogue, package and SMP2 C++ code generators together with a fully automated code merge facility, e) a dependency and build management approach that enables cross-platform large scale modular simulation developments with both source and binary deployments, and f) integration with the

Eclipse C++ development environment to allow seamless editing and debugging of C++ source code in a single coherent environment. This is described in further detail below.

Efficiency and Productivity for SMP2 Simulation Developments

One of the central objectives of the UMF v2 development was to support an "efficient and smooth" approach to software development for SMP2 simulations. Figure 2 below shows a number of UMF mechanisms and tools that support efficiency and productivity across the simulation lifecycle.

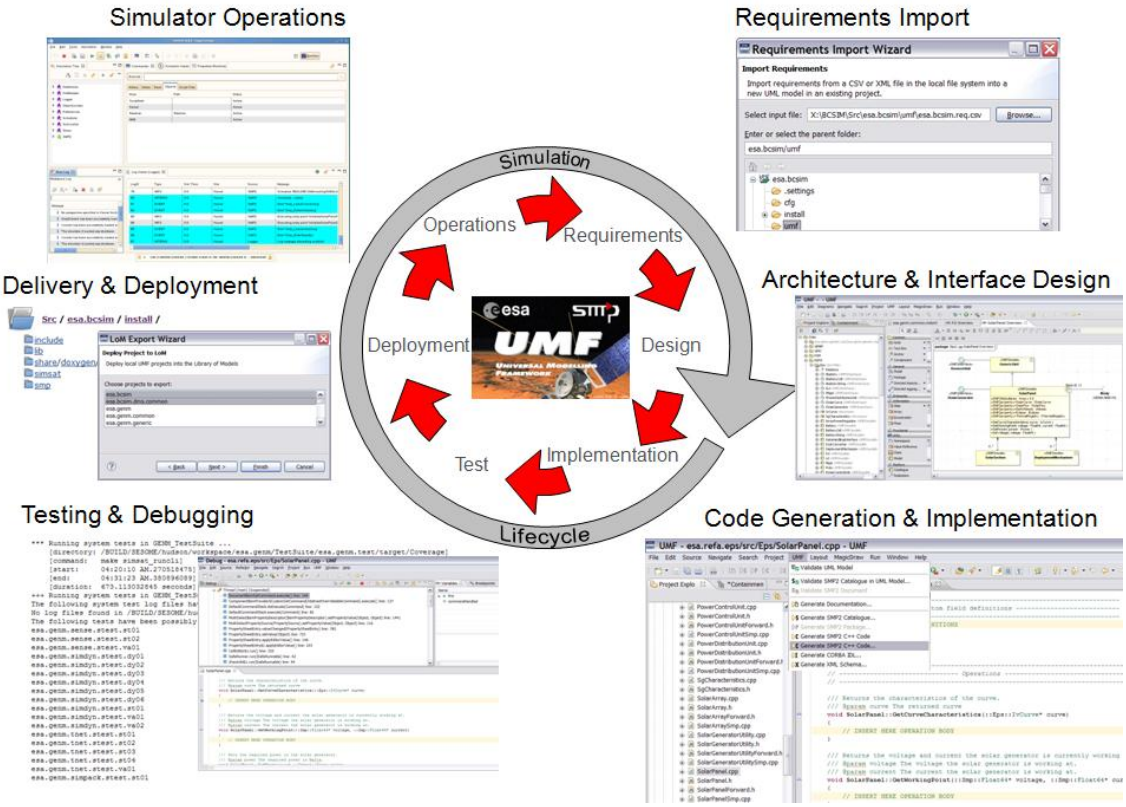

Fig. 2. UMF supporting all phases of the simulation lifecycle*: UMF plays a key role in improving usability and development efficiency across all phases of the simulation lifecycle*.

Requirements Import

Once an operational spacecraft simulator development is kicked off the first step for the development team typically consists in importing the requirements specified by the customer into the design tool. This enables to quickly set up the simulator design in UML.

Architecture and Interface Design

To ease the design activities UMF provides an integrated UML tool with a specific configuration that focuses the user interface on SMP2 concepts. The simulation design will typically make use of existing models, which is captured by entering appropriate dependencies.

Code Generation and Implementation

UMF provides a seamless path from the UML based design to an executable simulator via a comprehensive SMP2 generator tool chain:

- The **SMP2 Catalogue Generation Tool** generates catalogue(s) from UML.
- The **SMP2 Package Generation Tool** generates package(s) from catalogue(s).
- The **SMP2 Code Generation Tool** is a customisable template based C++ code generation engine that produces high quality SMP2 C++ code in line with SMP2 coding standards and based on the standardised SMP2 C++ mapping.

The developer enters the actual behavioural code in marked sections of the generated C++ code, where a customised Eclipse C++ Development Tools (CDT) based editor avoids unintended changes to generated code.

All tools may be executed stand-alone or in a chain via a **One Click** facility. In order to support iterative development and continuous design refinements, a change to the UML model is transferred to the code via a single developer action, whereby an integrated **Code Merger** ensures that changes in the UML model are seamlessly merged with existing hand-written sections in the C++ code.

Finally, UMF generates a **CMake based build system** with full integration into Eclipse CDT and a high level of configuration and customisation capabilities[1]. Apart from simply building the C++ code the build system also provides entry points for performing many development related tasks in an automated fashion, both from within the UMF UI and on command-line:

- make install: Populates the install folder with necessary artefacts for binary deployment
- make gencpp: Generate and merge C++ code from UML model (SMP2 C++)
- make genxsd: Generate XML Schema files from UML model (XML Schema)
- make gendoc: Generate ICD, SDD and/or SUM documentation (DocBook and DocX)
- make doxygen: Generate detailed design documentation (Doxygen)
- make test: Execute unit and integration tests (CppUnit)
- make memcheck: Execute unit and integration tests with memory leak analysis (Valgrind)
- make coverage: Generate coverage report after executing tests to determine code coverage
- make check: Check code for compliance with SMP2 standard (SMP Conformance Suite)
- make rules: Check code for compliance with coding standard rules (Logiscope Rulechecker)
- make audit: Compute code metrics and generate a report (Logiscope Audit)
- make simsat_deploy: Deploy models and (test) data to SIMSAT, including dependencies
- make simsat_runcli: Run system tests using SIMSAT

Apart from the above pre-defined tasks, developers may also create their own custom tasks, for example to support deployment to different SMP2 simulation run-time environments.

Testing and Debugging

While simulation models are being implemented it is of high importance to have testing and debugging facilities at the developer's finger tips. UMF provides a number of elements and mechanisms to support this:

- The **Unit and Integration Test Harness** is based on a small stand-alone SMP2 run-time environment and provides support for the well-known CppUnit test framework. Developers can create, execute and debug unit and integration tests directly within UMF.
- The **Debugging Facility** is based on the Eclipse C++ Debugger with GDB as back-end. UMF projects in the Eclipse workspace are configured such that the CDT debugger gets full visibility to the SMP2 C++ source code and associated include files, allowing the developer to take advantage of all features of the CDT debugger while debugging SMP2 models, both in stand-alone mode and when connected to the SMP2 simulation run-time environment, e.g. SIMSAT, BASILES or EuroSim.

Delivery and Deployment

An important step in the simulation development lifecycle is the delivery and deployment of the developed simulator to the customer. UMF supports this by a number of facilities and mechanisms:

- The **Document Generator** allows generating the bulk parts of ICD, SDD and SUM documents from the UML design in line with the ECSS E-40 engineering standards. The

[1] The CMake based build system of UMF is compatible with both Linux/GCC and Windows/MinGW environments.

generated document content is generated based on templates that can be fully customised to match project needs.

- The **LoM Packaging** mechanism allows creating binary or source packages for a UMF solution (see below). Packages can then either be deployed via the new LoM facility, or delivered via conventional file transfer such as FTP or CD/DVD media.

Integrated Validation

In day-to-day design and development work mistakes are not unusual, for various reasons. The development of an SMP2 based simulator based on UMF requires some learning curve, and even experienced developers sometimes do not know all aspects of the underlying technologies. Therefore, it is of high importance that mistakes are flagged to the developer as early as possible in order to minimise the impact to the overall development efficiency and quality.

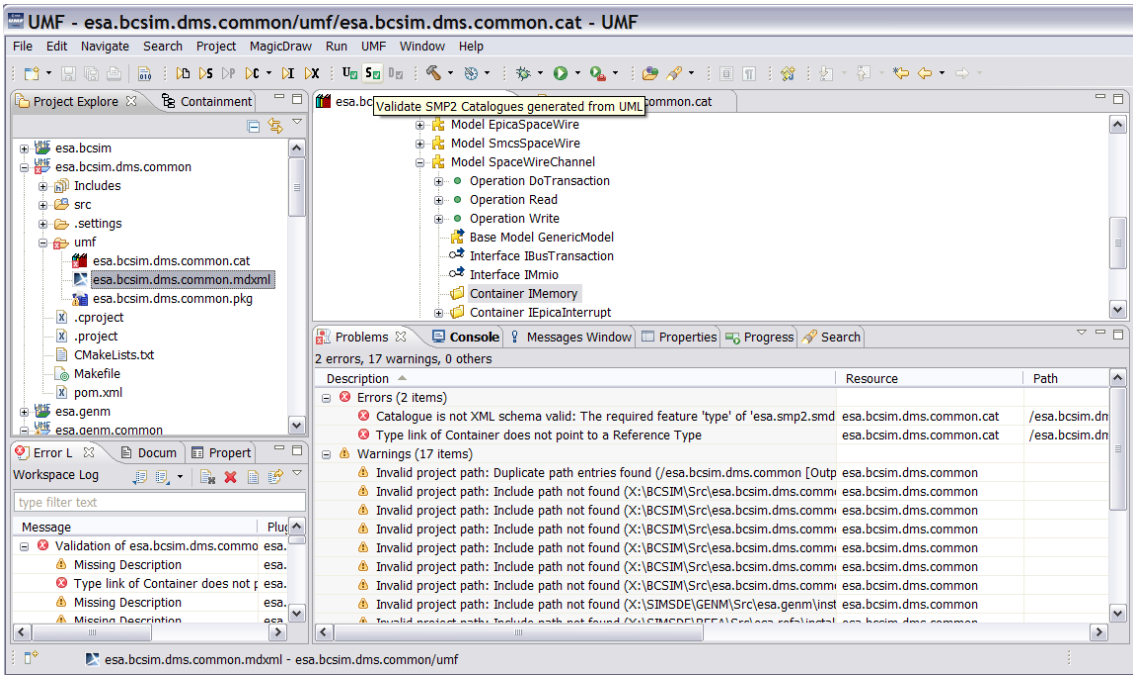

Fig. 3. UMF Validation: *Problems and inconsistencies are detected early, and a simply double-click allows to easily navigate to the source of the problem.*

To this end UMF provides a number of validation mechanisms that are fully integrated with the underlying tools:

- The **UML Model Validation Tool** detects mistakes in the semantics of the UML model. A double-click on the selected validation message in the Problems view directly navigates to the offending element in the UML tool.
- The **SMDL Validation Tools** detect mistakes or inconsistencies in SMP2 catalogues, packages, assemblies and schedules with respect to the SMP2 standard. Being based on the SMP Conformance Suite [4], the implementation takes into account the most comprehensive set of conformance rules to ensure full conformance with the SMP2 standard.
- A **Background Validation** facility ensures that validations are performed without distracting the developer. Validation results are collected into the Problems view, and appropriate markers are added to the associated files for seamless developer feedback.

Comprehensive Online Help

Last but not least, UMF enables a quick start and continuous support to the developer via an integrated and comprehensive online help. Help topics include for example the underlying UMF high-level concepts but as well detailed information for customising the UMF build system.

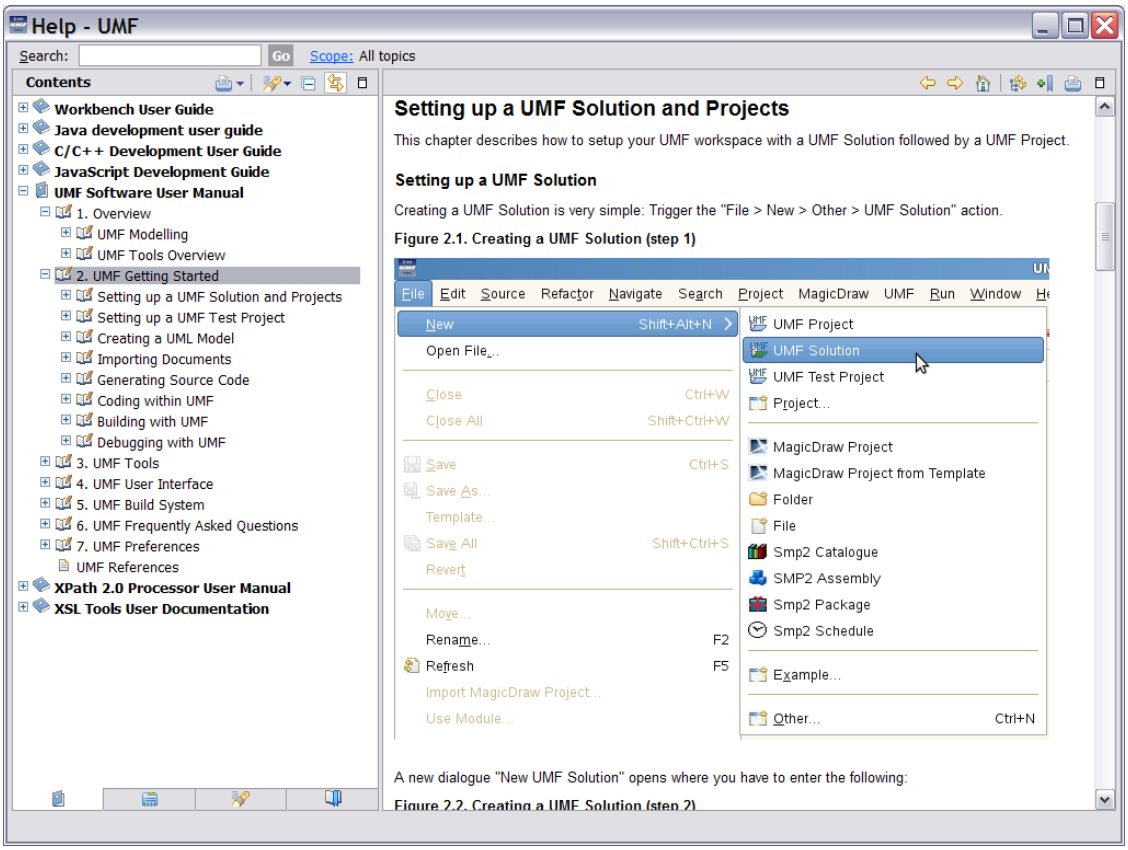

Fig. 4. UMF Online Help*: Comprehensive documentation at the developer's finger tips.*

Dependency Management via UMF Solutions and Projects

One of the major goals for the UMF v2 development is to achieve significant improvements in the overall simulation development and delivery process, especially with respect to modularisation of models and fostering of respective reuse, also in the context of the LoM facility. For this purpose UMF v2 provides a consistent and robust approach for dependency management between models.

As simulation models are typically grouped into a number of functional areas, UMF v2 introduces a new mechanism that enables the management of simulation models, where the focus is on supporting large scale simulation developments with hundreds of models and numerous dependencies between them. A two-layer packaging mechanism is provided in order to a) reduce the number of dependencies to be taken into account and b) to ease the way how a set of closely related models can be deployed and updated as a whole.

A *UMF solution* provides a top-level grouping mechanism and allows specifying dependencies to other solutions. A solution owns a set of *UMF projects*. The projects provide the actual design, implementation and configuration of the SMP2 models forming the solution, where a project always belongs to exactly one solution. On the other hand, a solution allows managing common dependencies or configurations for all its projects, such as shared requirements on design level or specific compiler flags for the build system.

Following the *convention over configuration* approach, the top-level directory layout within a solution or project is largely specified by UMF in order to a) provide full support for and integration with the Eclipse CDT and b) to enable packaging and deployment via the LoM facility.

UMF solutions form a dependency hierarchy which needs to be strictly organized as a tree. For example, in the context of an ESOC operational spacecraft simulator, the following solutions are typically involved (in bottom-up order of dependencies):

- *esa.smp2*: provides basic SMP2 mechanisms, e.g. the Model Development Kit (MDK)
- *esa.genm*: provides the ESOC Generic Models (GENM)
- *esa.refa*: provides the Spacecraft Simulator Reference Architecture (REFA)

CASE STUDY: THE BEPI COLOMBO SIMULATOR

Overview

The Bepi Colombo Simulator (BCSIM) project has been kicked off early this year. BCSIM is developed by an international consortium split across three geographies. An agile development process is adopted where frequent Sprint deliveries enable ESA to get early access to functionality which evolves in close alignment with the Bepi Colombo satellite development. Therefore, the following aspects are of pivotal importance to the BCSIM project in order to support the distributed team setup and associated agile process in a cost effective and efficient manner:

- Automated build and test of the simulator without the need for user interactions
- Automated creation of binary packages for deployment to the customer
- Break-down of the project into parts that can be assigned to different consortium members

Together with ESA the team has decided to base the development on the most recent SimSDE products: UMF v2 is used as baseline development environment, and REFA v2 and GENM v5 are used as baseline architecture and generic model libraries.

Feedback from the BCSIM team has already resulted to various UMF improvements, especially with respect to usability and performance that have a direct impact on day-to-day development work, but also related to the new binary deployment mechanism (LoM packaging).

Project Break-Down

BCSIM makes direct use of the UMF solution and project concept in order to capture dependencies between internal components and to CFI products, leading to the following dependency hierarchies:

- BCSIM source hierarchy: *esa.bcsim → esa.refa → esa.genm → esa.smp2*
- BCSIM test suite hierarchy: *esa.bcsim.test → esa.bcsim, esa.genm.test*

All implicit dependencies are also captured through UMF's automated dependency management, for example the BCSIM test suite solution *esa.bcsim.test* also depends on all dependencies of the *esa.bcsim* solution as well.

The *esa.bcsim* solution itself has been broken down into around 30 projects in line with typical REFA break-down but also to enable the assignment of functional areas to different consortium members. For example, the core Data Management System (DMS) is broken down into ten projects.

This setup worked well in practice and was easy to introduce and understand by all team members.

Automated Build and Test

In order to implement an efficient build, test and deployment mechanism, the BCSIM team has setup a number of automated Hudson [6] jobs that make direct use of underlying UMF mechanisms. For product assurance reasons both the debug and the release version of the simulator are built and tested every night. In case that all tests have passed a set of according binary packages is created – including the CFIs REFA, GENM and SIMULUS. When performing a Sprint delivery, these binary

packages are directly taken from Hudson and deployed to the customer's machines at ESA/ESOC. The following jobs are executed in Hudson, where the underlying UMF mechanisms are shown in brackets (see section *Code Generation and Implementation* above):

- Debug and quality assurance build stream:
  1. Build in debug mode and deploy to SIMSAT (make install, make simsat_deploy)
  2. Run automated unit, integration and system tests (make test, make simsat_runcli)
  3. Run quality analyses (make rules, make audit, make memcheck, make coverage)
- Release and delivery build stream:
  4. Build in release mode and deploy to SIMSAT (make install, make simsat_deploy)
  5. Run automated unit, integration and system tests (make test, make simsat_runcli)
  6. Generate documentation (make gendoc, make doxygen)
  7. Generate binary packages (mvn package)
  8. Create binary delivery as ISO files (SIMULUS, SIMSDE and BCSIM)

Binary Delivery and Deployment

The binary delivery (ISO files created via Hudson, see above) is directly installed at customer site based on an automated installation script developed by the BCSIM team. This ensures that all required CFIs are installed in a reproducible way, including SIMSAT, GROUND, SLEGM and the ESOC emulator from SIMULUS, REFA and GENM from SIMSDE and BCSIM itself.

To date, this delivery and deployment approach was successfully performed several times for Sprint deliveries, where the underlying mechanisms from UMF proved very valuable and stable.

CONCLUSIONS

A large step in terms of usability and tool integration has been made in UMF v2. However, practice shows that in a number of areas future improvements will still be required. For example, the overall turn-around time from a design change in UML to an associated C++ code update is a critical area for developer acceptance. Although this has been largely improved in UMF v2, the BCSIM team still experienced problems in a setup with a shared storage due to the high I/O load.

Apart from this, UMF v2 has been proven to provide a robust and productive development environment for SMP2 developments. Early user feedback has been continuously integrated throughout the SimSDE project. Being used by the Bepi Colombo operational simulator project as "launch customer" UMF v2 has already earned its merits in day-to-day development practice of a distributed development team.

REFERENCES

[1] SMP 2.0 Handbook, EGOS-SIM-GEN-TN-0099, 1.2, 2005-10-28

[2] EGOS-MF Release 1.0.5 SRN, EGOS-SDE-EMF-SRN-1001, 1.6, 2011-01-31

[3] UMF Release 1.1.0 SRN, EGOS-SIM-SIM-SRN-1001, 1.7, 2010-12-03

[4] E40-07 SMP-CS Release 1.0.1 SRN, SPB-SMPCS-874-SRN-001, 1.1, 2010-06-15

[5] Eclipse Indigo Platform: http://www.eclipse.org/downloads/packages/release/indigo/sr1

[6] Hudson: Extensible continuous integration server, http://hudson-ci.org