A FIRST EXPERIENCE ON USING SMP2 FOR THE VIRTUAL SPACE SYSTEM

Christian Laroque	VEGA
Mauro Pecchioli	ESA/ESOC
Nuno Sebastiao	ESA/ESOC

Presentation Outline

- Introduction to SMP2
- Virtual Space System Concepts
- Development Process
- Software Reuse
- GSTVi Assembly
- Events
- Dynamic Simulation
- Conclusion

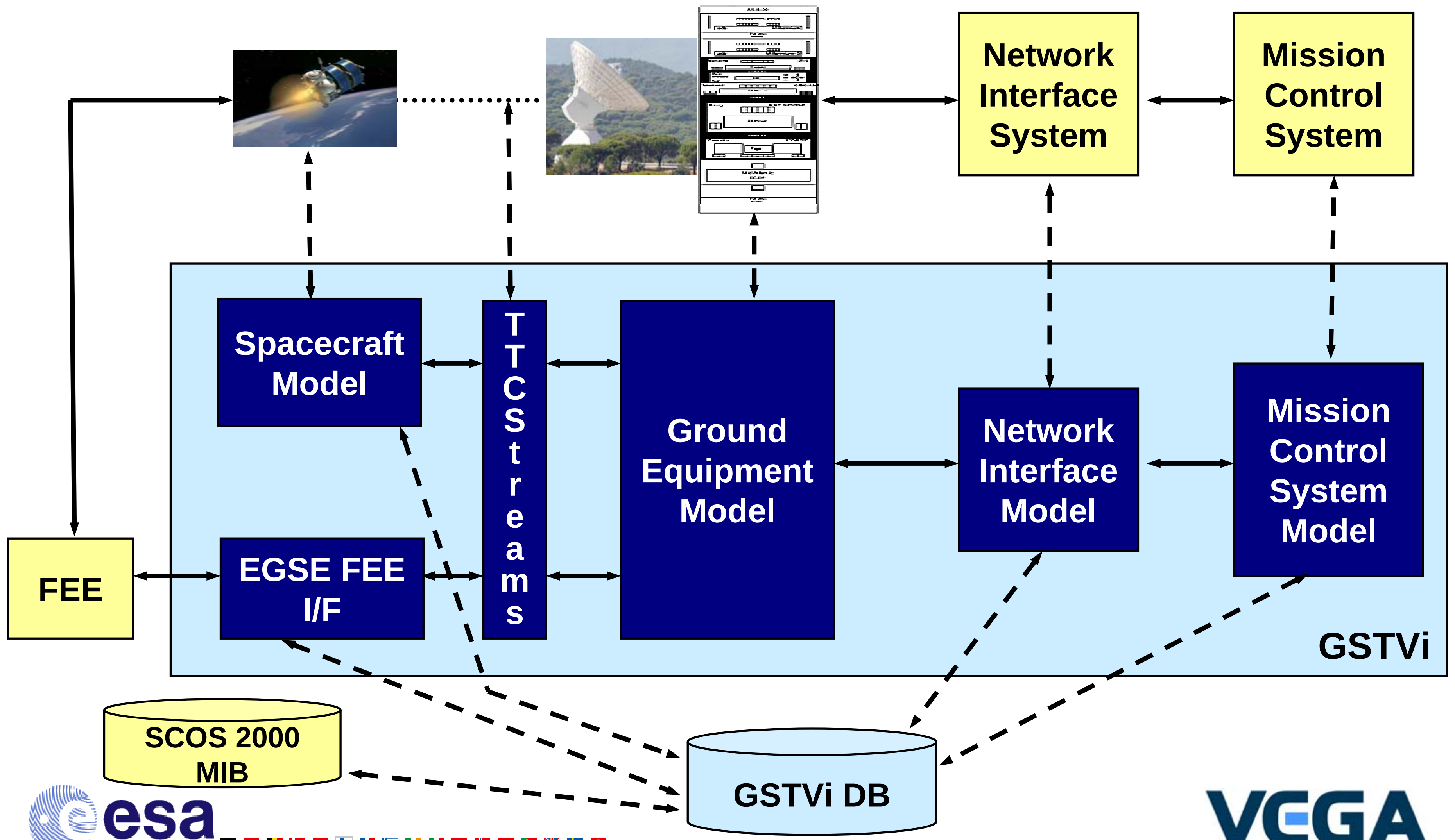
SMP2 Standard

The main purpose of the Simulation Model Portability (SMP) standard is to promote portability and reuse of simulation models. These goals are shared and highly regarded by both Space Agencies and Industry

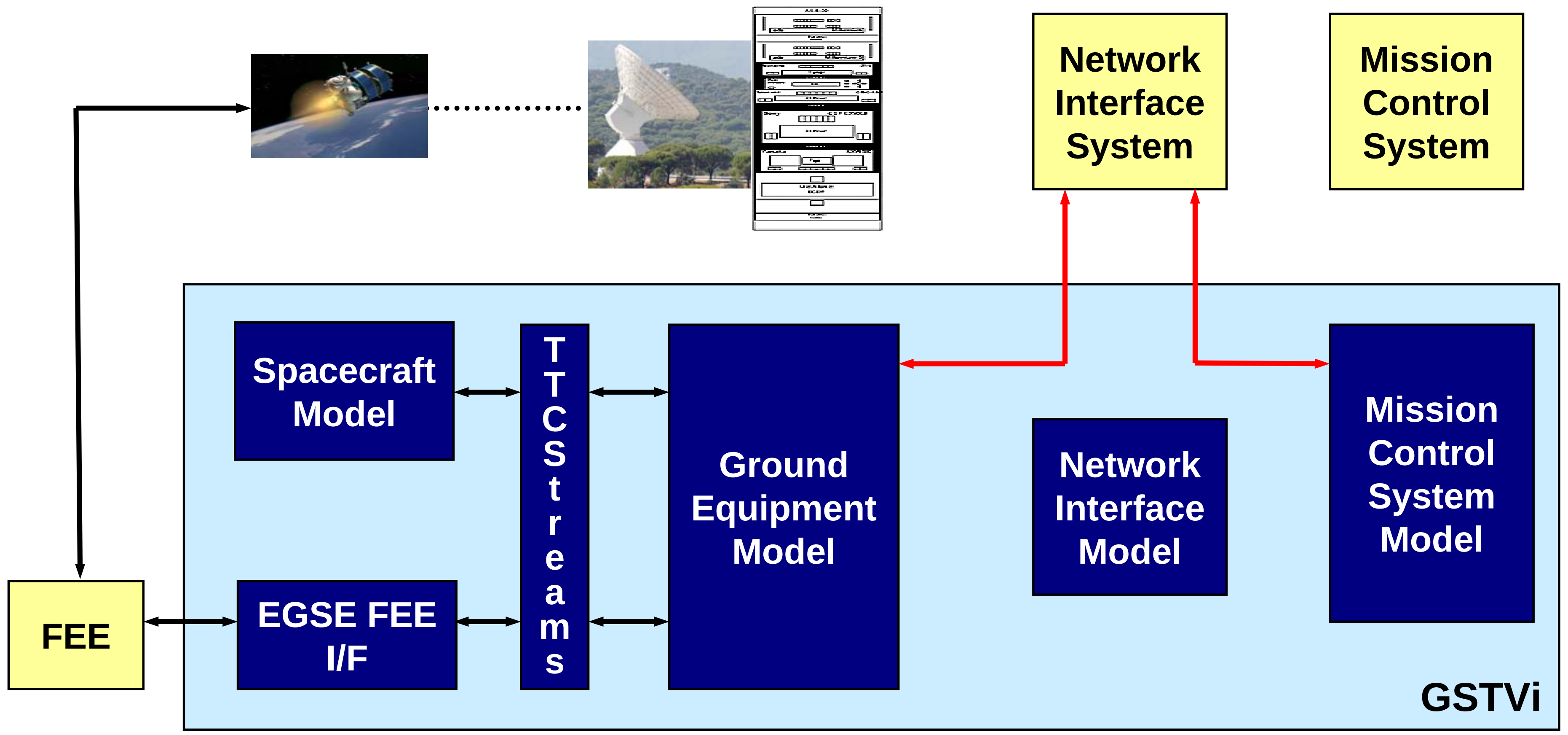
The new SMP2 standard, specified in the year 2004, offers the following advantages for the development of simulation systems:

- **Support for Modern Software Engineering Techniques**: use of modern software engineering techniques and in particular object-orientation and component-based design.
- **Model Reuse**: SMP2 improves the productivity and medium and long-term reliability in the development of simulators by helping the modeller in developing reusable models.
- **Model Integration**: SMP2 supports integration of individual models to form a complete system simulation. Improved model integration is closely related to model reuse.
- **Model Development Productivity**: SMP2 helps reducing the effort required by the model developer to implement the infrastructure elements of a model, such that it can be integrated into the simulation environment or with other models. The modeller can therefore focus on the development of the functional aspects of the model, and not on the infrastructure aspects.
- **Configurable and Flexible Simulation**: SMP2 allows developing highly configurable and flexible simulations, such that the simulated system configuration can be rapidly changed or evolved.

Virtual Space System Concepts (1)



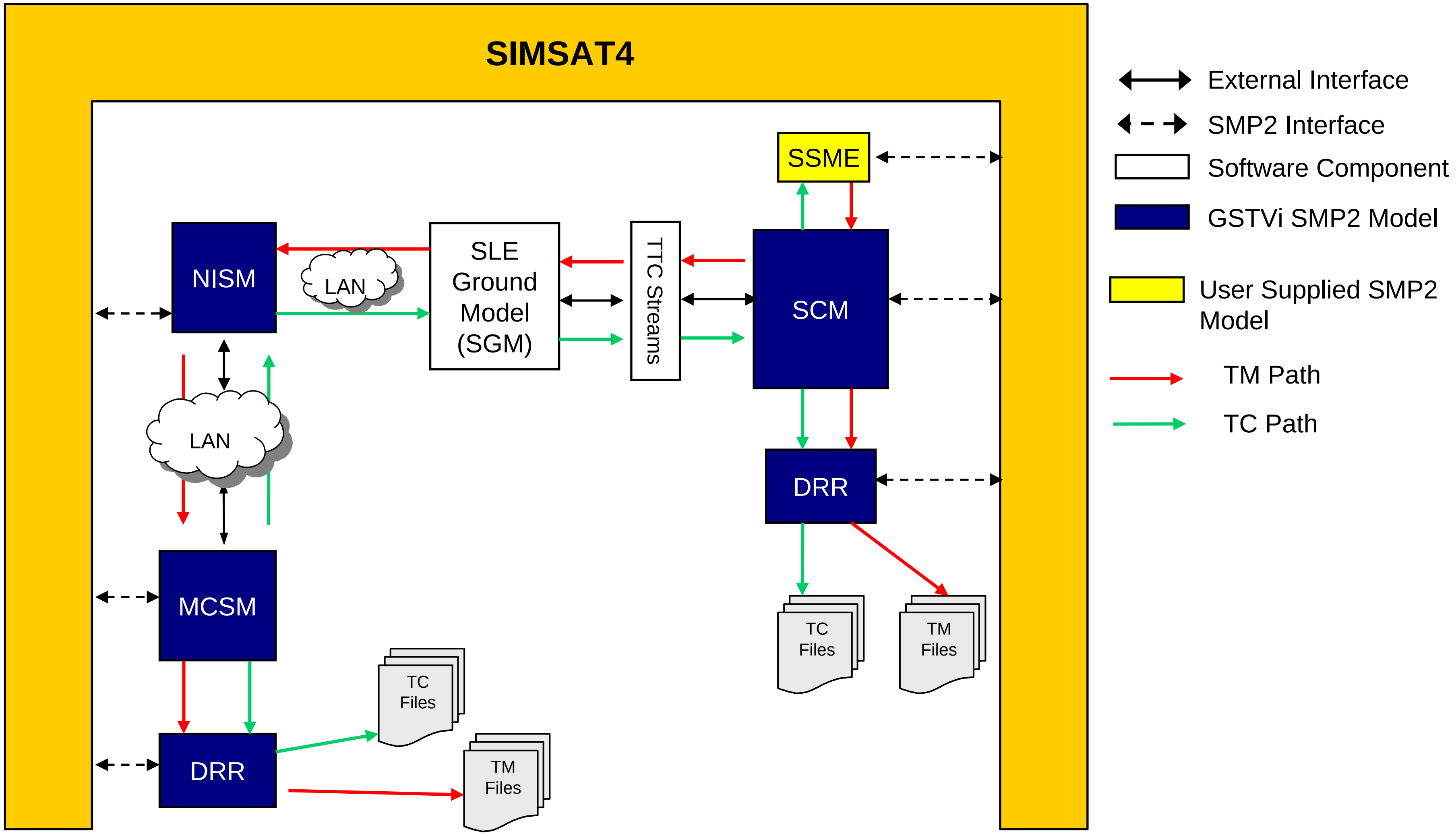
Virtual Space System Concepts (2)



Virtual Space System Concepts (3)

- GSTVi: Ground Systems Test and Validation Infrastructure
- GSTVi provides individual components that are assembled together to form a test system
- GSTVi provides predefined assemblies – new assemblies can easily be defined thanks to SMP2 standard mechanisms
- Assemblies correspond to 14 use cases that were selected to drive specification and design of GSTVi
- For each use case has been defined its objective, and the functionality to test
- GSTVi implemented against the version 1.2 of the SMP2 standard

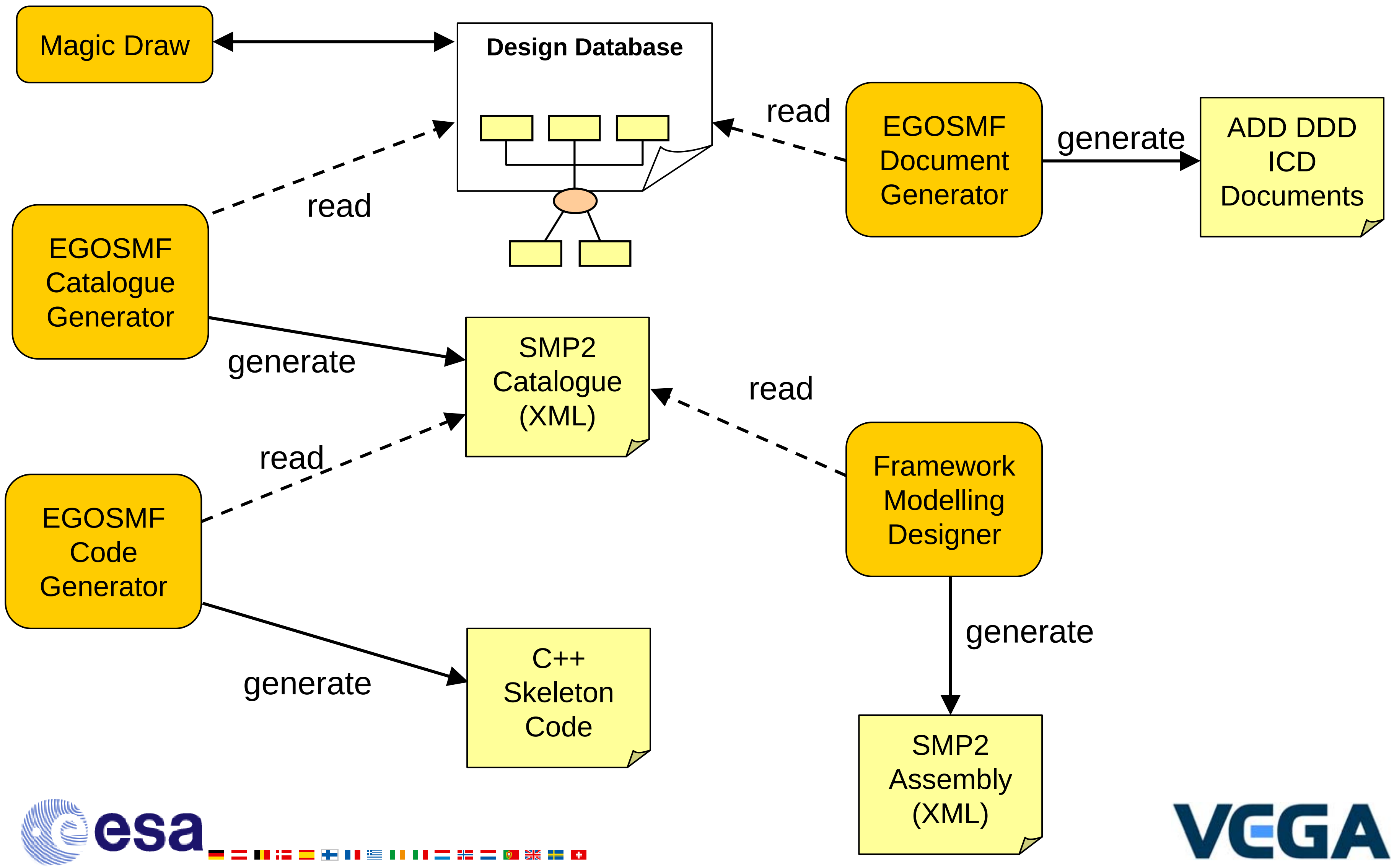
GSTVi Assembly Z



GSTVi

- GSTVi top level models:
  - Mission Control System model
  - Network Interface System model
  - Spacecraft Model
  - EGSE FEE I/F model
  - Runtime Configuration Manager
- GSTVi infrastructure models:
  - Parameter pool
  - TM/TC encoder decoder
  - PUS services
  - Data Recorder & Replayer (frames, packets)
- Performances:
  - Management of up to 50,000 parameters
  - Data rates up to 10 MBits/s

Development Process



Development Process

- Single source approach is a key issue

  → this is the only way to keep the project – UML design, documentation, source code, catalogues and assemblies - consistent

- Tools - EGOSMF
  - Used prototype tools for GSTVi
  - Some modifications / improvements have been implemented using the GSTVi experience and feedback
  - C++ code generator does not allow for all constructions
  - Concurrent development – how to share a UML model, SMP2 catalogues or assemblies

- Conclusion
  - Single source approach is highly recommended
  - Reliable Tools are very important
  - Tools must be easy to use and integrated in the development environment
  - Concurrent development must be carefully organised

Portability

- GSTVi was first developed using an enhanced version of SIMSAT3
- VEGA added to SIMSAT3:
  - A SMP2 adapter implementing the SMP2 standard and allowing loading and execution of SMP2 models in the simulation
  - Distribution of up to 150 models on 50 machines
- Development and integration was done on SIMSAT3
- When SIMSAT4 became available:
  - GSTVi was ported to SIMSAT4
  - No modification of the SMP2 model source code has been needed
  - Adaptation has been needed on the code loading the libraries, on the MMI and some infrastructure components (non SMP2)

→ SMP2 models are portable across platforms

Software Reuse

The GSTVi models are not implemented from scratch, but reuse lots of existing code:

- The Portable Spacecraft Simulator (PSS) for the implementation of the GSTVi spacecraft model, and the implementation of the IMBU interface;
- The NDIU Lite software for the implementation of the EGSE FEE I/F GSTVi model;
- The Hershel-Planck TM/TC Encoder and Decoder for the encoding and decoding of telemetry and telecommand packets in the GSTVi spacecraft and MCS models;
- The NCTRS for the implementation of the Network Interface Model (NIS).
- Some component from SIMPACK for the telemetry frame processor in the MCS model.
- Some component from the ROS/MEX simulator and Cryosat simulator for the implementation of PUS services on the GSTVI spacecraft model:
  - Memory management (PUS service 6)
  - On-board storage and retrieval (PUS service 15)

Software Reuse – Migration to SMP2

- Reused software must be migrated to SMP2
- Two approaches have been identified

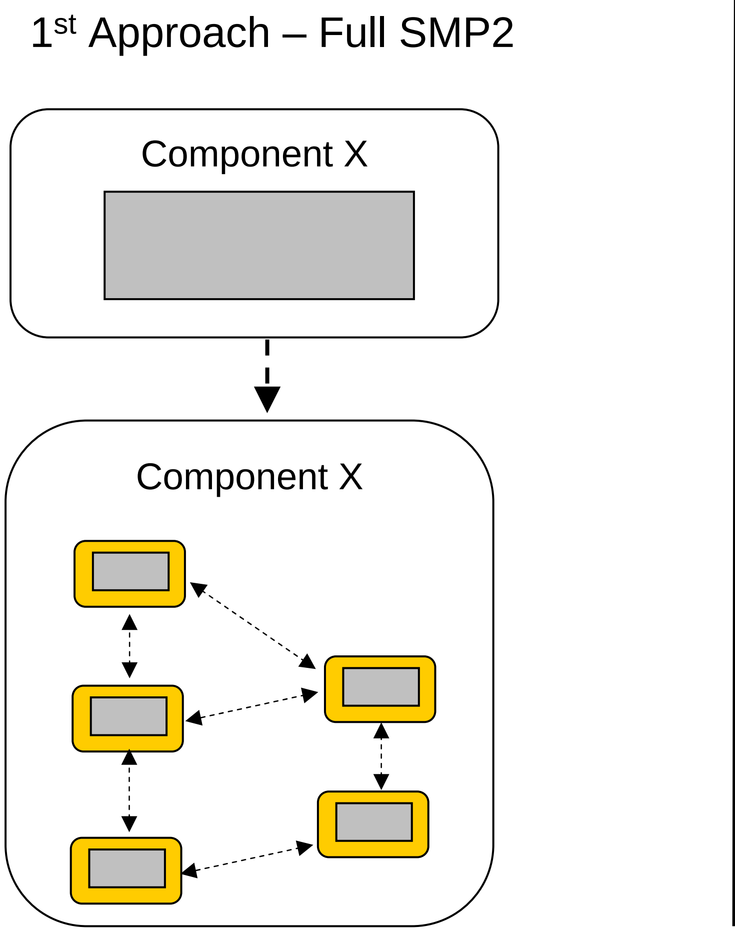
  - 1st Approach - Full SMP2

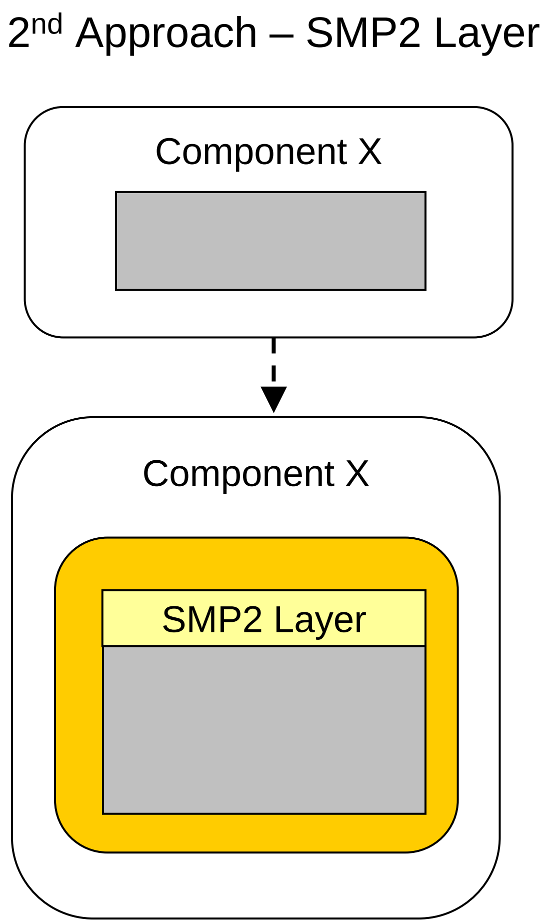
    → Fully migrate the existing code to be SMP2 compliant

  - 2nd Approach – SMP2 layer

    → Keep the existing code as is, and provide a SMP2 layer around this code to allow interfacing it with the rest of the simulation SMP2 code.

Software Reuse – Migration to SMP2





🟨 SMP2 Models ▨ Existing Component Implementation

Software Reuse – Migration to SMP2

- 1st Approach - Full SMP2
  1. Reverse engineer the existing code in the UML model
  2. Clean-up the model, change the types to use SMP2 native types
  3. Define SMP2 native types for external non-SMP2 software
  4. Split the existing components in small SMP2 models that can be assembled using the SMP2 assembly
  5. Define the fields and operations that must be published to the simulation
  6. Generate the catalogues files
  7. Generate the code from the catalogue files, and insert in the generated code skeleton the code from the existing implementation
  8. Compile and test the SMP2 models

  Example: GSTVi spacecraft model PUS services, parameter pool

- 2nd Approach - SMP2 Layer
  1. Design a SMP2 layer around the existing code
  2. Generate the catalogue file containing the SMP2 layer
  3. Implement the SMP2 layer

  Example: GSTVi NIS model – need to keep existing core implementation, and SLE API interfaces

Software Reuse – Migration to SMP2

	1st Approach: Full SMP2	2nd Approach: SMP2 layer
Advantages	▪ The component is fully SMP2, and can take full advantage of all SMP2 functions. ▪ Only one environment development needs to be available, i.e. the SMP2 compliant one with associated tools. ▪ The code is uniform and easier to maintain and understand.	▪ The existing code can be re-used with a reduced effort and risk. ▪ There is no need to re-test all component functions, since most of them remain un-changed.
Drawbacks	▪ The modification of the existing code to be fully SMP2 compliant requires a big effort. ▪ Requires full re-testing.	▪ To interface the SMP2 code to the existing non-SMP2 one, additional code needs to be written which only purpose is to convert types/interfaces and to publish data. ▪ The code is not uniform. For instance, SMP2 code uses SMP2 types, the non-SMP2 code not. ▪ The development environment is not uniform across one component.

Software Reuse – Migration to SMP2

Conclusion:

- Careful analysis must be performed during the design phase to choose the correct approach
- Interfacing SMP2 code to non SMP2 code can be a challenge
  - Identify classes to migrate to SMP2
  - Convert types to SMP2 types
  - Convert interfaces to SMP2 ones where possible
- Powerful C++ code generator tools are needed in order to allow generation of classes representing SMP2 models, but to which can be added and maintained non-SMP2 code

GSTVi Assembly (1)

- The SMP2 assembly
  - Defines how a collection of SMP2 model instances is assembled for the runtime simulation
  - Offers the possibility to initialise SMP2 model fields
  - → Very powerful mechanism for assembling the GSTVi components in a test environment
- But
  - SMP2 assembly does not provide all needed functions for GSTVi
  - → No possibility to automatically execute a script when an assembly is loaded – this would be useful to load catalogues files or shared libraries of specific versions
  - → No possibility to link an SMP2 assembly with the SIMSAT architecture file defining the components loaded in the simulation at runtime
- Definition of GSTVi assembly, which is composed of
  - A SMP2 assembly file;
  - A SIMSAT architecture file;
  - A script file used to load the GSTVi model libraries or run a specific setup test.
- Users only need to select the GSTVi assembly at runtime to fully initialise the simulation
- GSTVi defines 14 different assemblies

GSTVi Assembly (2)

- When the user selects the GSTVi assembly, the GSTVi system:
  1. Checks the GSTVi assembly
  2. Starts a simulation with the specified architecture file
  3. Executes the start-up function defined in the script file
  4. The start-up function:
     - Loads the catalogues
     - Loads all the needed GSTVi model libraries;
  5. Loads the SMP2 assembly
  6. Executes the setup function defined in the script file to perform any final initialisation step.

Events

- SMP2 provides mechanism for
  - Definition of events
  - Firing events
  - Performing specific processing on event reception
- But one important missing function is that there is no way to associate data with the event
- For GSTVi, there is a need to associate data with the events:
  - Associate the event with the name of a telemetry parameter for the event “parameter value goes out-of-range”, “parameter value goes in-range”, “parameter value matches a predefined value”
  - Associate the event with the name of a telemetry packet and the packet itself for the event “TM packet generated”
  - Associate the event with the name of a telecommand packet and the packet itself for the event “TC packet received”

→ GSTVi has defined its own scriptable-event mechanisms

Loading and Publication of Sub-Models (1)

GSTVi has specific needs:

- GSTVi has „static“ SMP2 models defined in the assembly
- But most of the models are created dynamically at initialisation time after reading the GSTVi mission tailoring database.
  - Example: The TM/TC parameters are defined in configuration, and can only be created after reading the configuration
- One GSTVi mission tailoring database is defined for each mission to support
- The choice of the mission is done at initialisation by the user, after the simulation is started

→ No possibility to define this statically in the assembly !

- The user may decide to switch to another mission during the simulation to simulate spacecraft passes

→ Need to unload the models and load new ones

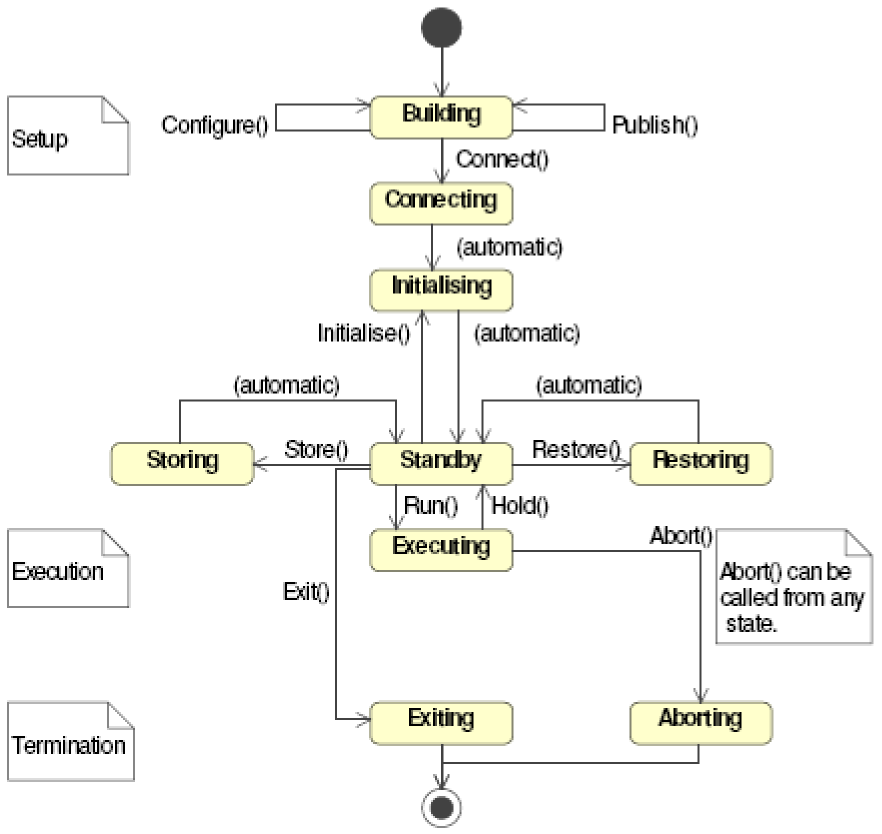
Loading and Publication of Sub-Models

The initialisation of the simulation must be done in 4 separate steps:

1. Start the simulation and load the SMP2 adapter
2. Perform the first initialisation (Configure()) which allows publication of some data and operation to the simulation
3. Allow the user to select the GSTVi mission tailoring database
4. Terminate the initialisation process (Connect()) and switch to Standby state.

→ To achieve this, the SMP2 standard had to be modified (from version 1.1 to 1.2) on request of GSTVi specific requirements

Simulation state diagram



Loading and Publication of Sub-Models

The SMP2 standard imposes important constraints on the design and implementation of the SMP2 models :

- Publication of SMP2 models and fields and operations is only possible in the Building state
- Published data and models cannot be un-published and un-loaded

→ This is fine for modelling of static models, which do not create any model or data during the simulation

→ But this is an important limitation for modelling of systems comporting dynamically created models/data, like GSTVi

Examples:

- re-selection of a different mission
- management of a command stack on the MCS model
- management of SLE links on the NIS model

Conclusion (1)

- The SMP2 standard offers the needed component model function needed for the development of the system
- SMP2 Assembly is very useful to assemble the GSTVi virtual space system
- SMP2 allows to dynamically create models and publish dynamic data in the initialisation phase
- The SMP2 models are easily portable – GSTVi was originally developed on SIMSAT3 and the models have been easily ported to SIMSAT4 without the need to change any code.
- Single source approach for design and implementation is efficient if well used and controlled
- Tools offer a good productivity if reliable and well integrated in the development environment
- Concurrent development must be analysed and controlled, depending on the available tools.
- Reuse of existing code is not always simple, and can be a challenge

Conclusion (2)

- The SMP2 component model misses some function
  - AddRef() and Release(), memory allocator
- SMP2 does not address how dynamic shared library implementing the SMP2 models are loaded. No SMP2 interface for loading shared library is provided
- Once in stand-by state, it is not possible to dynamically load and create models or to publish additional data
- No possibility to un-load data or models
- No possibility to associate data with events
- SMP2 containers are mapped in C++ to a vector, with no possibility to decide on a key for indexing the vector
  - → performance problem for large containers

Conclusion (3)

Support for Modern Software Engineering Techniques	✓
Model Reuse	n/a
Model Integration	✓
Configurable and Flexible Simulation	✓
Model Development Productivity	✓

VEGA

www.vega-group.com

Independent Programme and System Assurance
Technical Excellence **.** Pragmatic Solutions **.** Proven Delivery