EGS-CC Phase B - a Report

Martin Götzelmann

Telespazio VEGA Deutschland GmbH, Darmstadt, 64293, Germany,
martin,goetzelmann@telespzio-vega.de

ABSTRACT

The European Ground Systems – Common Core (EGS-CC) is a European initiative to develop a common infrastructure to support space systems monitoring and control in pre- and post-launch phases for all mission types. This is expected to bring a number of benefits, such as the seamless transition from spacecraft Assembly, Integration and Testing (AIT) to mission operations, reduce cost and risk, support the modernisation of legacy systems and promote the exchange of ancillary implementations across organizations. The initiative is being performed as a collaboration between ESA, European National Agencies and European Industry. A presentation of the initiative and its current status can be found in the paper "EGSC-CC: the Initiative is Becoming a Reality" [1]. This paper provides a report on the Software Requirements Engineering and Architectural Design, Phase B of the EGS-CC project, which has been performed by a geographically distributed industrial team in close collaboration with the EGS-CC System Engineering Team (SET) .

INTRODUCTION

Phase B of the EGS-CC project covered Software Requirements Engineering and the Architecture and Interface Design and was performed by Telespazio VEGA, CGI, GTD, Terma, and Dutch Space under ESA Contract. It started in March 2013 and ended in June 2014 with closeout of the Preliminary Design Review.

Specification and design of EGS-CC faced the following main challenges:

- In order to be widely accepted and used as monitoring and control infrastructure by space operators and space industry, EGS-CC must support a large variety of systems and operations contexts. This implies that it must on the one hand provide a rich set of generic commonly usable features and on the other hand exhibit a high degree of extensibility and adaptability to specific operational environments. The EGS-CC SET has therefore defined ambitious design goals including e.g.
    - Open, component based, service oriented architecture;
    - Native support to automation at all levels;
    - High performance and scalability;
    - Extensibility via binary interfaces;
    - Long term maintainability.
- The selection of the technology to be used for EGS-CC was performed by a separate activity concurrently with the design; the architectural design itself should be technology neutral as far as possible while taking maximum advantage of the features provided by the recommended technology;
- The EGS-CC programme has a large number of stake holders and the success of the project heavily depends on continuous involvement of all parties to ensure that their intentions and needs are met.

In the following sections this paper will describe the approaches adopted in Phase B to meet these challenges:

- Modular layered architecture based on rigorously specified services provided by replaceable and extensible components;
- Model Based Development: the complete specification and design is implemented in a UML model adopting a formal approach that supports verification of the design and lays the foundation for model based development in the following phase;
- Consideration of testing and in particular test automation as an essential design aspect from the outset;
- Iterative design process with frequent workshops with and reviews of intermediate results by the stake holders and continuous coordination with the technology selection project.

LAYERED EGS-CC ARCHITECTURE

Overview

The EGS-CC product is conceived as an infrastructure package that can be used for the development of monitoring and control systems for spacecraft including Mission Control Systems (MCS) and Electrical Ground Support Equipment (EGSE) for spacecraft AIT. This can be better explained by stating what EGS-CC is and what it is not.

- EGS-CC is *not* a monitoring and control system.
- EGS-CC is *not* just a collection of infrastructure components that can be integrated into a monitoring and control system.
- EGS-CC is
    - a platform on which a monitoring and control systems can be built and which provides core monitoring and control features as well as application support functionality;
    - a set of components that allow adapting the core monitoring and control features to the operation environment;
    - a test framework that can be used to validate EGS-CC itself and systems based on EGS-CC.

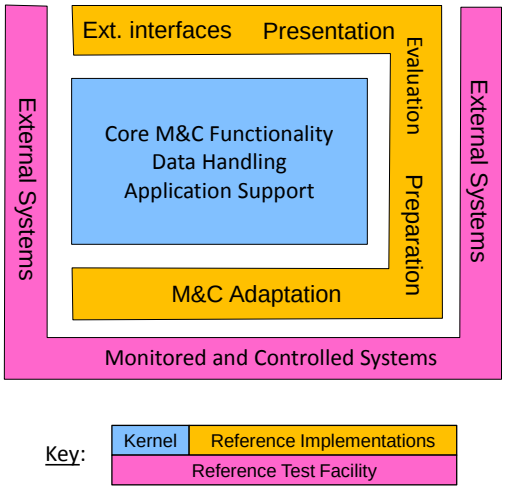

Fig. 1. Main EGS-CC Architectural Layers

These considerations are reflected by the top level architectural layers of EGS-CC shown in Fig. 1.

The *EGS-CC Kernel* provides core monitoring and control functionality in a manner that is independent of the interfaces to monitored and controlled systems and the packaging of the data received and transmitted. In addition, the Kernel includes data handling and general application support features. According to the long term maintenance and evolution policy defined by the SET, the Kernel will always be delivered as a single package which cannot be modified by system developers using it but shall enable extensions for specific needs.

The *EGS-CC Reference Implementations (RI)* provide functionality required to adapt the Kernel to a specific environment in which it is operated covering interfaces to monitored and controlled systems, pre-processing of raw monitor data, and encoding and formatting of control information, as well as interfaces to external systems based on the CCSDS Recommendation for Mission Operation Services [5]. In addition, this layer includes reference implementations for features that some prospective users of EGS-CC prefer to provide by other implementations either because those implementations cover a wider scope of applications or are more sophisticated. Examples of such features include user interfaces, data evaluation and post-processing, and preparation tools. The Reference Implementations are therefore designed as a set of components that may be replaced individually by alternative implementations without any impact on other components.

The *EGS-CC Reference Test Facility (RTF)* is a test environment delivered as part of the EGS-CC product. It is foreseen to be used for validation of the EGS-CC product but also for systems based on EGS-CC and includes simulators for EGS-CC interfaces as well as test management and test automation features. As for the Reference Implementations components the RTF must support easy replacement of components and integration of additional simulators.

Each of these architectural layers present specific design challenges which are described in the following sections together with the design approaches adopted in response.

EGS-CC Kernel

Functional Scope and Design Challenges

The functional scope of the Kernel has been the main driver for the decomposition into high level components as shown in Fig. 2. These components can be grouped into three layers:

- Core monitoring and control functionality that is independent of specific interfaces and protocols including
    - Processing of monitoring information once decoded from transfer containers and presented by standard data types;
    - Processing of control actions (referred to as activities) to the extent possible independent of specific interfaces and protocols;
    - Management of monitoring and control data definitions;
    - Archiving of monitoring and control information; and
    - Automation.
- Data access, distribution, and archiving services for processed data (M&C Access API) and source data (Source Data Access). This layer also provides access to EGS-CC Kernel services for external systems via the service integration platform.
- General infrastructure, application support, and runtime management, provided to all EGS-CC components.

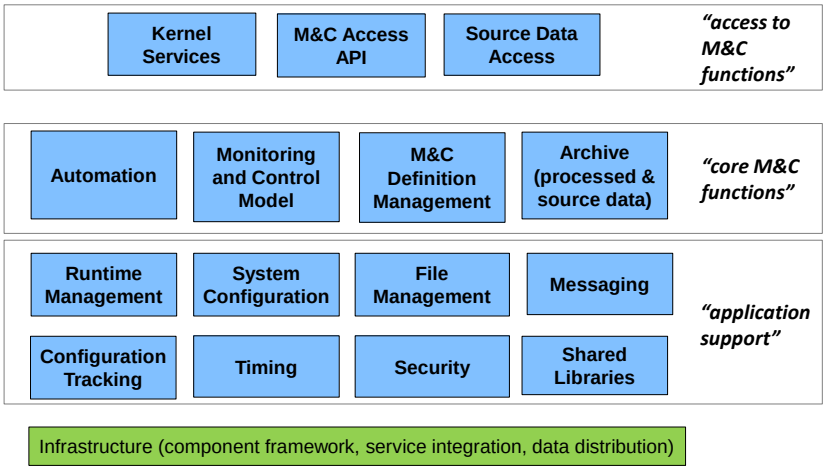

Fig. 2. Decomposition of the EGS-CC Kernel

The specific characteristics of the Kernel impose some specific design challenges:

Finding the right abstractions: Because the Kernel components shall be independent of any specific interfaces and shall be capable of supporting a variety of different operation contexts it is essential to define concepts at the right level of abstraction and to defend this level of abstraction against the natural tendency to mix them with interface specific concepts and data structures. Many of the EGS-CC concepts had already been conceived in Phase A of the project and have been elaborated and refined during the design process while others have been developed within phase B. Examples of such concepts are described later in this section.

Defining services from a client perspective. One of the design goals of EGS-CC is to provide a service oriented architecture and this approach has been systematically applied. It has been observed however, that initial versions of the service definitions were very much driven by what a component has to offer rather than systematically analysing usage scenarios and optimizing the design from a client's perspective. Through iterative review and revision the service design could be very much improved in this respect.

Enabling customization of the kernel. As mentioned earlier the long term maintenance and evolution policy for EGS-CC defines that the Kernel is always delivered as one package and cannot be modified by developers using EGS-CC to build a system. This essentially means that interfaces exposed by the Kernel shall remain stable but there is no guarantee that interfaces not exposed but defined only between Kernel components, let alone component internal interfaces, remain unchanged from one version to the next. Nevertheless there will be the need to adapt the kernel to the operational context and to adjust or extend its functionality. Customisation of the Kernel for a given system is supported by the following capabilities.

- *Configuration*: all components provide extensive configuration options. Editing and offline management of configuration data is supported by a (replaceable) Configuration (Data) Management component of the Reference Implementations and online distribution and management of configuration data is supported by a dedicated System Configuration Kernel component.
- *Tailoring* refers to the definition of monitoring and control data items and their organization to adapt the system to a specific application/mission. Tailoring in this sense includes the preparation of automation procedures used to automate testing or operation of a spacecraft.
- *Scripting* can be used to extend the functionality of the software. Scripts can be executed within the automation component (see later in this section) and at system level to automate configuration and launching of applications.
- As a final line of defence, the EGS-CC design supports numerous extension points at which developers can attach software extensions implementing mission specific algorithms or functions to complement the functionality available from the EGS-CC Kernel.

All of these customization features are available also for components of the Reference Implementations. Those components, however, can additionally be completely replaced individually by custom implementations.

Monitoring and Control Model

A central concept of EGS-CC is the Monitoring and Control Model (MCM), which is extensively covered by [1] and therefore only discussed in outline here with focus on design relevant aspects.

The MCM is a model of the space system which is monitored and controlled for test (in an EGSE system) or for operational purposes (in an MCS). The MCM enables the capture of all relevant information related to the Space System in a structured way which reflects the functional decomposition of the space system itself. The knowledge held by the MCM is of two sorts: static and dynamic. The static knowledge encompasses all monitoring and control data produced during the development and the maintenance of the Space System and is used to tailor an EGS-CC system for a given mission. This static knowledge is based on the principles of the M&C view of the Space System Model (SSM)

defined in [2]. It is referred to as the "*MCM Definitions*". The dynamic knowledge encompasses all monitoring and control data produced by the Space System during testing and operations (e.g. the content of all sent TCs and all received TM). This is referred to as the "*MCM State"*.

The MCM concept ensures a clean separation between the M&C abstract view and generic processing and the specific processing related to the data units exchanged with the controlled systems. This approach allows the application of the same M&C kernel to different types of controlled systems, such as spacecraft, EGSE systems, ground station equipment, etc. This means that the M&C system can monitor and control not only the target system (e.g. a spacecraft), but also all other contributing ground systems (e.g. EGSE supporting equipment, and the EGS-CC itself), independent of the format in which data are transferred.

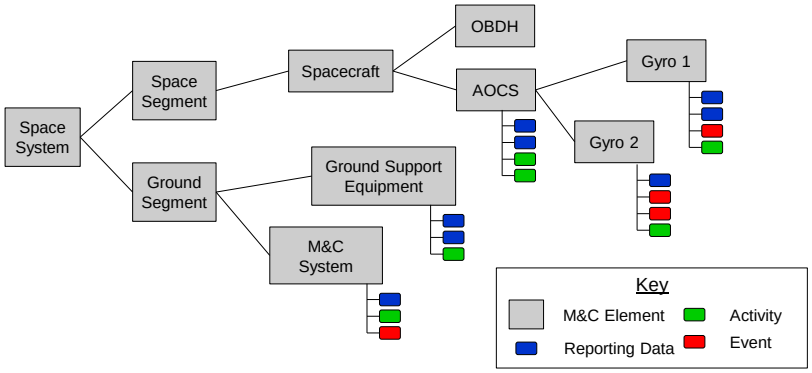

Fig. 3. Monitoring and Control Model Concept

The MCM contains a hierarchy of Monitoring and Control Elements (MCE). Monitoring and Control Elements typically correspond to the elements of the space system decomposed from the monitoring and control perspective, providing the operator with the desired level of abstraction. Each MCE may include parameters describing the state of the represented space element, events, and activities. The concept is illustrated by Fig. 3; the concept of activities is discussed to more detail in the following section.

Activities and Automation

An activity is an abstraction of a control function invoked through the MCM that can be implemented as a command either to the space segment or to the ground segment, an automation procedure, a script, or any software function provided by EGS-CC. Fig. 4 illustrates the concept from an MCM perspective.

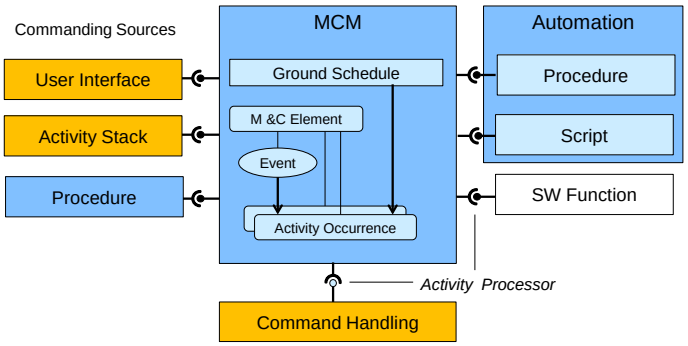

Fig. 4. Activities and Automation

An activity defined by tailoring may be invoked by a commanding source, which may be a human user (via the user interface), an automation procedure, or any other SW component. The figure shows the Activity Stack, a special Reference Implementation component, capable of implementing sequences of potentially interlocked act ivy invocations prepared e.g. by a mission planning system. When an activity is invoked by a commanding source, the MCM creates an activity occurrence which is subsequently used to monitor the execution of the activity. From the perspective of the MCM, an activity occurrence is executed by a component that implements the Activity Processor interface. The component providing this interface is also responsible for defining what verification stages will be supported to monitor execution of the activity and to provide the associated reports for verification stage updates. Any component can implement this interface and register it with the MCM; the interface to be selected for a given activity is defined by an associated Service Access Point, which may be overwritten by the commanding source, in order to have the activity occurrence executed by an alternative activity processor, e.g. for testing purposes.

Activity invocations by command sources can be for immediate execution or may specify an invocation time. In the latter case the activity occurrence is loaded to a ground schedule which will release it for further processing at the specified time. Finally activities can be associated with events such that those activities will be executed whenever the specified event occurs.

Access to Monitoring and Control Data

Essential services provided by the Kernel deal with storage, retrieval, and distribution of monitoring data. A major constraint for the Kernel design is that it must have no dependency on the specific operations context including also the protocols and data structures by which data are exchanged with controlled systems. Therefore a clear distinction is made between M&C data processed by the MCM and therefore known to the Kernel and "source data" holding data as received from controlled systems.

Source data are considered opaque data annotated with information attached by components of the Reference Implementations that receive and pre-process the data. Source data are stored to a source data archive and the annotations are used as keys for data retrieval. The Source Data Access (SDA) component provides services by which data producing components can forward data for storage and online distribution and data consuming clients can subscribe to data based on the annotations.

The term processed data refers to the output of the Monitoring and Control Model. These objects are stored to Processes data Archive and can be retrieved using object properties and the archive knows their structure. The M&C Access API (MCA) component provides services by which clients can subscribe to processed data using versatile filters and update specifications including cyclic, changes only, and various combinations of these.

The services provided by the SDA and the MCA components provide "one stop shop" features allowing clients access to data independent of the location at which data are stored and the independent of the time at which data are processed. Subscriptions to data may define a time period which starts in past and extends into the future. In such a case data will be initially retrieved from the Archive and delivery of the data will eventually seamlessly switch to live data. Both components support a timely delivery mode in which data may be dropped in favour of delivering always the most recent data and a complete delivery mode that ensures data delivered are complete. In the latter case, delivery may switch between live data and archived data if the client cannot take the data at the speed live data are produced. The M&C Access API also provides control services e.g. to invoke activities. This feature hides the complexity of a potentially distributed MCM.

Session Management

A monitoring and control system session is a logical grouping of EGS-CC components dedicated to the processing of the data of a controlled system and generating a separate set of outputs distributed to clients and stored to the processed data archive. Several system sessions can be operating concurrently, e.g. dealing with test of different subsystems. System sessions are associated with a set of M&C definitions and configuration data referred to as System Operation Baselines (SOB). The components operating within a system session have access to dedicated repositories for archiving ("data space"), files ("file space") software configuration ("configuration space") and log messages ("log book"). Data in the configuration space are part of the SOB as are a subset of the files in the session "file space". Finally each system session runs within a dedicated time frame, which can be ground system time (e.g. UTC) or simulated time based on an offset and optionally a scale factor applied to ground system time. These concepts are illustrated in Fig. 5.

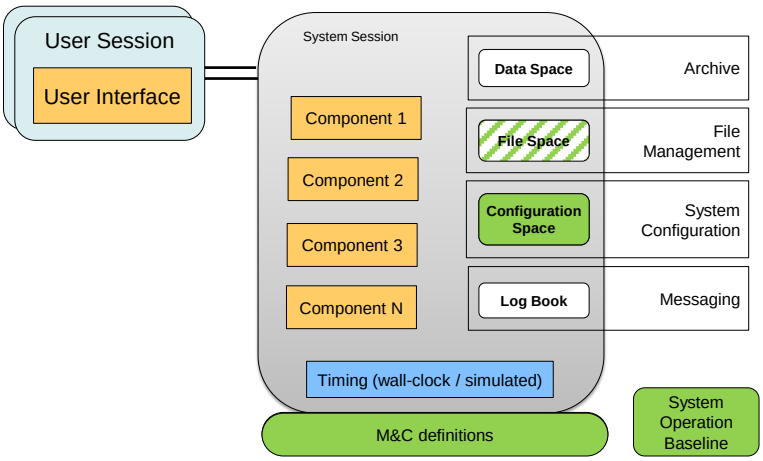

Fig. 5. EGS-CC System Session Concept

As shown in the figure, a "user session" is created for each user logging into an EGS-CC based system. After logging in, users must select the service session they wish to connect to.

System sessions are managed by the Run-time Management (RUM) component based on session profiles that identify the application instances required for a given session as well as the System Operation Baseline. To start a session, RUM will create the repositories, populate them with the required data, and launch all application instances defined in the profile. RUM itself as well as the components handling the repositories (Archive, File Management, System Configuration, and Messaging) are not part of any system session as the data they manage must be preserved also after a system terminates and must be accessible also outside a system session.

EGS-CC Reference Implementations

Functional Scope and Design Challenges

The functional scope of the Reference Implementations covers

- Adaptation of the Kernel to the monitoring and control context, i.e.:
  - Provision of interfaces to monitored and controlled systems;
  - Pre-processing of monitored data to the extent that this depends on specific protocols and data formats;
  - Processing and verification of commands sent to controlled systems;
  - Modelling of M&C services such as the ECSS Packet Utilisation Services (PUS) [4];
- User Interfaces;
- Default preparation tools for automation procedures, user defined display, and configuration data ;
- Basic post-processing tools;
- Special applications and interfaces not required for all deployments.

As pointed out earlier, an essential difference between the Kernel and the Reference Implementations is that RI components may be replaced completely by a bespoke implementation without affecting any other component. This requirement presents a number of specific design challenges:

Finding the right granularity: In order to make replacement of components by mission specific implementation effective, the decomposition of the Reference Implementation into replaceable components is essential. The decomposition described later in this section is the result of careful analysis of the processing steps to identify coherent functionality that might have to be replaced for a given mission.

Supporting generalised processing concepts and flexible interfaces: In order to enable use of EGS-CC in a large variety of different operation contexts a number of generalised processing concepts have been defined including for instance support for transmission routes that may require different means for command transmission and execution verification (see [1]). Obviously the design must support those concepts which implies in particular use of flexible design patterns for the interfaces between the components to accommodate passing information that my vary according to the route on which commands are transmitted and monitoring data are received.

Designing for the unknown: An essential feature of the Reference Implementations design is that it must aim at easy integration of interfaces and features that are not fully defined at the time of development. The characteristics and the scope of such interfaces and features have been derived from EGSE and MCS currently in use but because of the large variety of different approaches, protocols, and data structures suitable abstractions need to be defined. The design responds to this challenge with the concept of generic component specifications, which define an interface handler and describe how it must be integrated with other components of the Reference Implementations but make no assumptions on how the interface handler will connect to and interact with the equipment to which it interfaces. To connect external equipment to an EGS-CC based system, developers of that system will have to design and implement an interface handler that also implements the appropriate EGS-CC generic component specification. EGS-CC Reference Implementations may or may not provide an implementation of a generic component. This design approach has been applied for the following type of interfaces:

- *Telemetry and Telecommand Data Interfaces* that inject CCSDS telemetry frames into EGS-CC and accept command data at the level of CCSDS Frames or CCSDS Packets. For this interface EGS-CC provides a specialised implementation that supports the CCSDS Space Link Extension (SLE) standard [3].
- *Packet Data Interfaces*, that can support any packet or message based interface receiving monitoring packets and sending command packets. The packets may be CCSDS Space Packets but could also be messages of any kind that can be constructed and decoded according to a tailored packet structure definition. A typical specialisation for such a specification is the interface to a SCOE (Special Check-out Equipment) in the context of spacecraft AIT.
- *Cryptographic Services* used to interface mission specific implementations to secure telemetry and telecommand data. Cryptographic services may be provided “in loop” i.e. only to secure command packets or frames or decrypt telemetry frames or packets. Alternatively an implementation may cut the processing loop in that it includes the complete processing of commands after the interception point and/or of telemetry before the interception point.

Monitoring Chain

The top-level decomposition of the monitoring chain based on the principles described in the previous section is shown in Fig. 6 in a slightly simplified manner. The figure also shows how the components of the monitoring chain interact with the Kernel components.

Interfaces for reception of monitoring data are defined as generic component specifications for which an interface/protocol specific implementation must be provided by developers of an EGS-CC based system. The only

interface implementation already included in the Reference Implementations supports the CCSDS SLE services. Processing in the monitoring chain is based on the concept that all data to be processed further are passed to the Source Data Access (SDA) component of the Kernel for distribution and optionally for storage to the Source Data Archive. Processing components will subscribe to SDA for the data of interest. Identification and decoding of packets is supported by a set of generic services implemented by a Packet Encoding and Decoding (PED) component based on tailored packet structure definitions. These packet definitions are part of the M&C definitions stored and maintained by the Kernel.

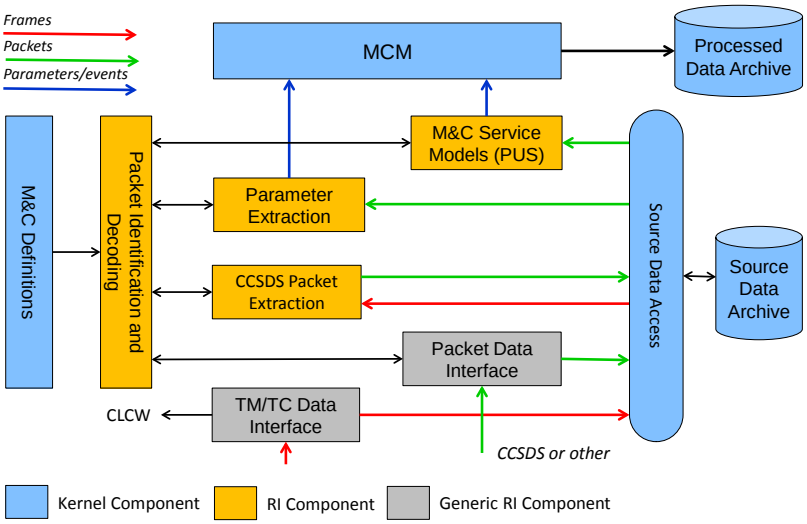

Fig. 6. Top Level Design of the Monitoring Chain (simplified)

For processing of CCSDS Frames the Reference Implementations include a component for extraction, identification, and pre-processing of packets. As a final step the Parameter Extraction (PEX) component extracts parameters from packets and injects them into the MCM for processing and storage into the Processed Data Archive. Beside basic packet handling and parameter extraction, the Reference implementations include a set of M&C Service Models (MSM) providing more advanced processing features such as loading commands on a schedule in the controlled system. While the services and interfaces provided by the MSM component are generic the only services for which models are specified and implemented are those defined by the ECSS Packet Utilisation Standard [4].

Commanding Chain

Fig. 7 shows the top-level decomposition of the commanding chain and the interaction of the components implementing the chain with Kernel components. As explained earlier, sending of commands to controlled systems is one of several options by which an activity can be implemented and therefore, components that interface to the MCM for processing of commands implement the Activity Processer interface; these interfaces are marked by "(AP)" in the figure. The Activity Processor interface allows passing the activity occurrence ID and a set of arguments that are used to encode the command packet using the Packet Encoding service provided by the same component that also provides the packet decoding service. In addition to the activity occurrence, the Activity processor interface allows passing the command route that determines the components to be used for subsequent processing steps as well as a set of "processing directives" to control individual processing steps, where applicable.

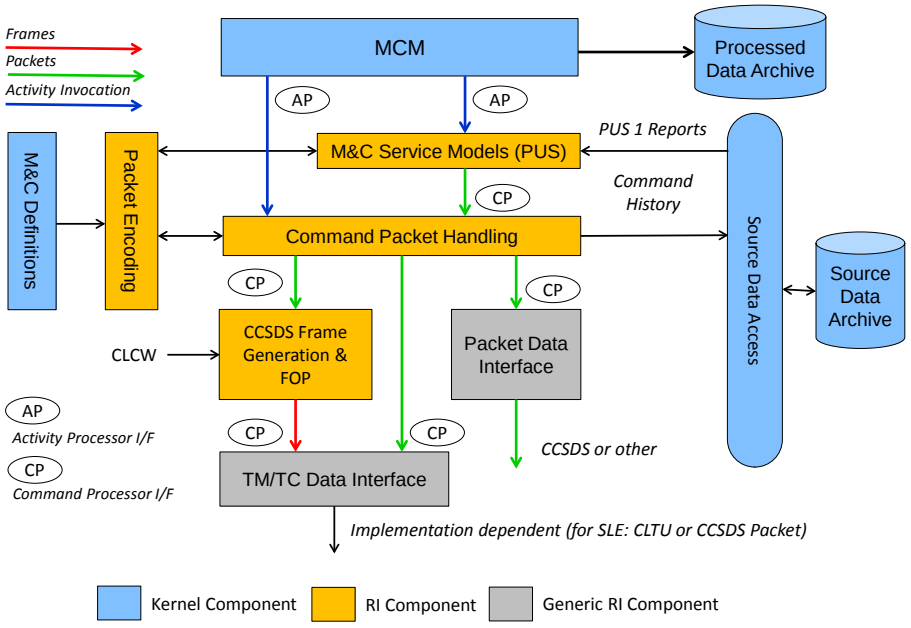

Fig. 7. Top Level Design of the Commanding Chain (simplified)

Components implementing further processing steps all implement a generic Command Processor interface, marked by "(CP)" in the figure. This interface allows passing an opaque command structure as well as the activity occurrence ID, the command route, and the processing directives. Components implementing the interface will take into account those processing

directives they understand and ignore others and will forward the command as defined by the command route. The final processing step consists of sending the command to the controlled system using the appropriate protocol which must be performed by an implementation of either the Generic TM/TC Data Interface or the Generic Packet Data Interface specification. As for the monitoring chain, the Reference Implementations include an implementation of the Generic TM/TC Data Interface for the CCSDS SLE services.

Verification of command transmission and execution by the controlled system may also depend on the transmission route. For instance, uplink of a telecommand via a ground station will be monitored by a set of uplink verification reports whereas such reports might not be supported by a SCOE. Therefore each Command Processor interface is associated with a Command Verification Reporter interface, by which the component announces what verification stages it supports and subsequently submits verification reports. In the same way, the Activity Processor Interface is associated with an Activity Reporter interface which allows announcing verification stages and passing verification reports. For controlled systems supporting the PUS Standard [4], the verification stage reports related to on-board acceptance and execution are provided by the PUS Service 1 model dealing with command verification packets. This model provides a generic command verification service such that the PUS-1 implementation can be replaced by any other verification service if needed.

A number of M&C Service Models maintain a model of the on-board service which is progressed according to commands sent. In some cases reports can be requested to be downlinked and can then be compared with the model state to support synchronisation. An example of such a model is the onboard schedule model (PUS 11) that shows the commands currently loaded on the schedule and their status. In EGS-CC those models are maintained in the MCM as a hierarchy of M&C elements, parameters, events, and activities such that they can be viewed and processed as any other monitoring information. Service requests would typically be modelled as activities associated with the model. For the invocation of such service requests the M&C Service Models also implement the Activity Processor interface. When an activity is invoked the models will generate the appropriate commands using the Packet Encoding service and will update the service model in the MCM accordingly.

EGS-CC Reference Test Facility

The third element of the EGS-CC product, after the Kernel and Reference Implementations, is the Reference Test Facility (RTF). The RTF is not part of an EGS-CC system, but is a testing framework. It supports system-level testing of the Kernel and of the Reference Implementations for EGS-CC product validation; it supports system testing of complete EGS-CC systems; and it supports testing of externally supplied components in integration with EGS-CC. The last of these cases allows components developed by external entities to be integrated and tested in parallel with ongoing EGS-CC development and maintenance.

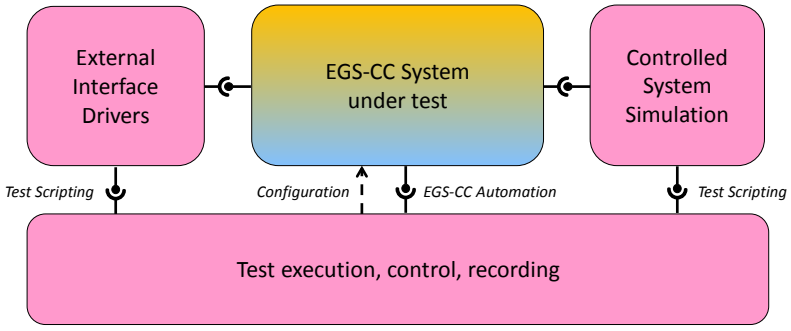

Fig. 8. Reference Test Facility

The RTF therefore simulates the operating environment of an EGS-CC system. It can also run tests in which part of the actual operating environment is integrated. In this way the elements of a ground segment can be tested together in different combinations, with the missing elements simulated by the RTF. The constituent elements of the RTF and their interaction with EGS-CC are depicted by Fig. 8.

COMPONENT BASED, SERVICE ORIENTED, AND MODEL DRIVEN DESIGN

Following the directives of the EGS-CC System Engineering Team, the design of EGS-CC strictly applies the following principles

Component based design: EGS-CC is conceived as a set of components supported by a component framework that can be used to build monitoring and control systems. Components are closed in the sense that they expose well defined interfaces and services but hide the internal structure and implementation. As such components can be easily replaced by implementations that support the interfaces and behaviour defined by the component specification. The system decomposition into components is recursive: the EGS-CC design refers to components resulting from the first level decomposition as Level 0 Components or L0 Components for short. An L0 component is further decomposed into L1 components, and so on until the level of implementation classes has been reached.

L0 Components of the Reference Implementations must be easily replaceable without impact on any other component. This requirement asks for more than just replacing the software by a different implementation. In EGS-CC a component is a self-standing replaceable unit also in the sense of documentation, configuration, tailoring, and testing. Therefore the complete documentation including Software Requirements, Interface Design, Configuration, Tailoring, and Deployment is provided as a specific Software Specification and Design Document (SSDD) for each L0 Component. The SSDD includes all information that is required by developers of an EGS-CC based system. The internal design of L0 Components, i.e. the decomposition into L1 components, delegation of L0 Component interfaces and the interfaces between L1 components is not included in the SSDD but provided in the UML model. For review purposes, a set of hyperlinked HTML files documenting the UML model can be generated and distributed.

For better structuring, L0 Components are grouped into subsystems and subsystem level information is compiled in a subsystem SSDD. The decomposition logic and the associated documentation are illustrated in Fig. 9.

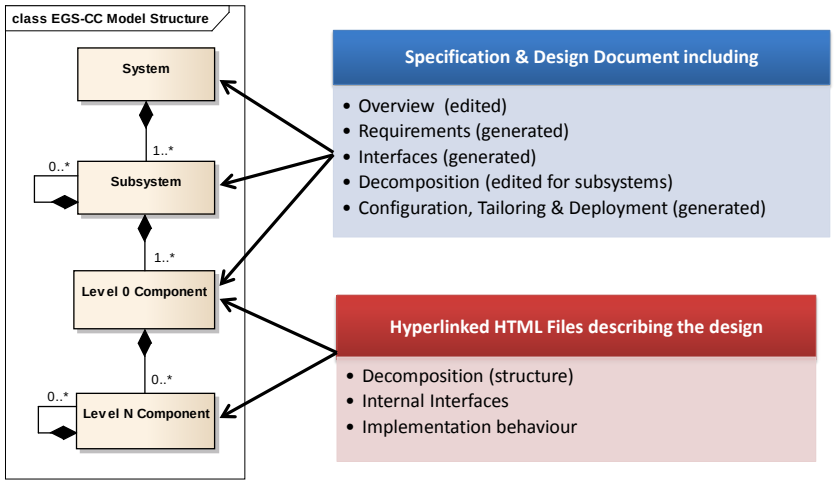

Fig. 9. Decomposition Logic and Associated Documentation

Service oriented architecture: All interfaces between components and interfaces to external systems are strictly modelled as services using design patterns based on the SoaML standard [6]. EGS-CC components specify services, the interfaces used for the services and service roles, including the service provider and the service consumer. Component specifications prescribe that a component shall provide a service by inheriting the service provider role specification. Likewise inheritance of a service consumer role specification defines that the component will use a given service. With this approach, components are fully decoupled and only depend of the service specifications. Linking of component interfaces is expected to be implemented by the component framework in a manner that is transparent for application code. It is recognised that this may require bespoke extensions to third party products based on EGS-CC specific conventions.

Model driven development: The complete design has been developed using the Unified Modelling Language in such a way that will allow further refinement of the model up to generation of source code from the model. In addition to all design aspects both user requirements and software requirements have been entered into the model such that tracing can be performed and verified within the model as well. The documentation of the software requirements and of the design has been generated from the design model to a very large extent.

Technology Independence: The architectural design as documented in the UML model and the SSDDs does not assume use of any specific technologies. Interaction between components that depends on infrastructure technologies such as the component framework has been modelled using design patterns based on SOAML. On the other hand technologies that are suitable for use by EGS-CC have been subject to analysis in a separate project which has built a proof of concept prototype primarily to evaluate the performance of candidate technologies. Based on the results of that project an initial technology baseline has been defined by the Phase B project, which will have to be further elaborated in the following phases. The feasibility of implementing the design patterns used for EGS-CC with the technologies included into the baseline has been verified by prototyping.

As a final precaution, the EGS-CC Kernel architecture includes a special component “Infrastructure” foreseen to cover any add-on development that might be required to extend the capabilities of selected third party products to meet EGS-CC requirements.

PHASE B ENGINEERING APPROACH

Beside the technical challenges described in the previous sections, the EGS-CC Phase B project had to cope with organisational challenges including:

- The requirement to involve a large number of stakeholders into the specification and design process in order to ensure that their needs are properly addressed by the project results;
- A team including members from four different companies distributed over Europe;
- A very tight schedule with no tolerance for delays.

The approach taken in response to these challenges is depicted in Fig. 10. The project has been split into 3 engineering iterations each focussing on one of the EGS-CC architectural layers (Kernel, Reference Implementations, and Reference Test Facility) and addressing Software Requirements and Component Design concurrently.

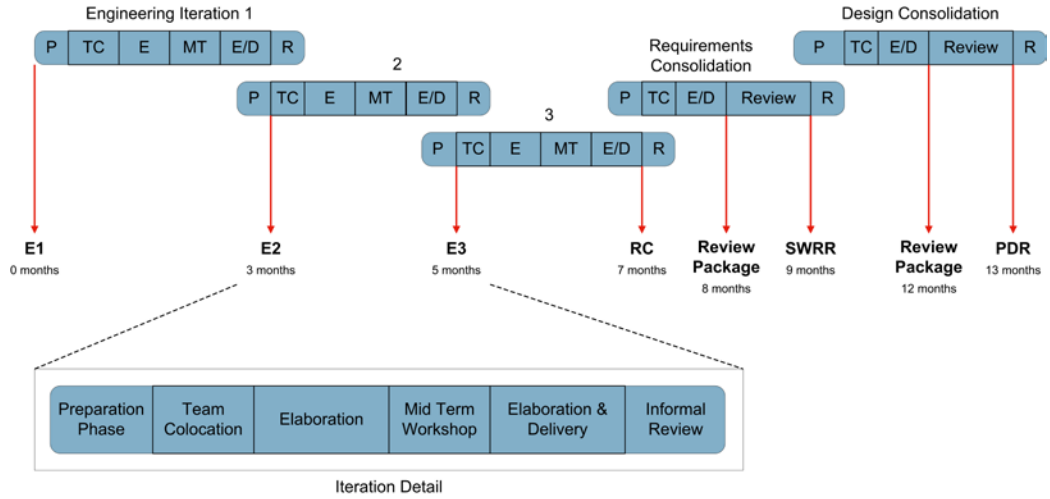

Fig. 10. EGS-CC Phase B Project Plan

Each iteration started with a preparation phase, followed by a team colocation session of a full week and subsequent elaboration of the results by individual team members. In the middle of each iteration, a workshop with all stakeholders was organised to present the results achieved and discuss open questions. The conclusions of the workshop were used to update the specifications and the design. The iterations were then concluded by an end iteration delivery followed by an informal review of the delivery by the stakeholders. Updates resulting from those reviews were implemented and delivered at the end of the following iteration. The three engineering iterations were followed by a consolidation phase for the software requirements specification terminated by a formal Software Requirements Review (SWRR) and a consolidation phase for the design terminated by a formal Preliminary Design Review (PDR).

Concurrent engineering of requirements and design, extended team colocation sessions, and frequent reviews of intermediate results have proven to be very effective and has significantly contributed to increase of consensus and quality of the product. Maintaining the schedule without any slippage has not been easy and required very hard work but at the end this goal has been achieved as well.

REFERENCES

[1] Pecchioli, M. and Carranza, J.M., EGSC-CC: the Initiative is becoming a Reality, Workshop on Simulation for European Space Programmes (SESP), 2015

[2] ECSS-E-ST-70-31C, Space engineering – Ground systems and operations – Monitoring and control data definition, July 2008.

[3] CCSDS-910.3-G-3, Cross Support Concept - Part 1: Space Link Extension, Green Book, Issue 3, March 2006

[4] ECSS-E-ST-70-41A, Space engineering – Ground systems and operations – Telemetry and telecommand packet utilisation, January 2003

[5] CCSDS-520.0-G-3, Mission Operations Services Concept, Green Book, Issue 3, December 2010

[6] Service oriented architecture Modeling Language (SoaML), Issue 1.0., Object Management Group, May 2012