EVOLUTION OF THE SMP2 STANDARD INTO ECSS SMP

Peter Fritzen(1), Alberto Ingenito(1), Robert Blommestijn(2), Vemund Reggestad (3), Anthony Walsh(3)

(1)Telespazio VEGA Deutschland GmbH
Europaplatz 5, 64293 Darmstadt, Germany
Email: peter.fritzen@telespazio-vega.de, alberto.ingenito@telespazio-vega.de

(2) European Space Research and Technology Centre
Keplerlaan 1, 2201 AZ Noordwijk, Netherlands
Email: robert.blommestijn@esa.int

(3)European Space Operations Centre
Robert-Bosch-Str. 5, 64293 Darmstadt, Germany
Email: vemund.reggestad@esa.int, anthony.walsh@esa.int

ABSTRACT

The latest version of the Simulation Model Portability specification version 2 (SMP2), as currently implemented in many simulation environments, and in use in most of the recent space missions, has been released in 2005. In 2009, an ECSS Technical Memorandum (E-TM-40-07) for a “Simulation Modelling Platform (SMP)” was developed by ESA with the participation and contribution of the European space community. Based on experience gained in the use of SMP2 and E-TM-40-07, an ECSS Steering Group was established in early 2013, comprising of representatives from industry and institutional stake-holders involved in the development of E-TM-40-07, with the objective to make the Level 1 Conformance of SMP a full ECSS E40 series level 3 document, based on a consolidation of the current TM.

While it was an objective to maintain backwards compatibility with the existing, well-established SMP2 specification, lessons learned in more than 10 years have led to proposed changes of different nature which include:

1. Clarification of ambiguities left in the SMP2 specification, by better documentation and strict requirements;
2. Introduction of additional interfaces for features not supported in SMP2;
3. Rationalisation of existing interfaces, and alignment with C++ 11.

These changes will be implemented in a draft version of an ECSS SMP E40 series level 3 document, and will undergo a public review.

This paper shall summarise the delta between SMP2 and E-TM-40-07 and the proposed changes between E-TM-40-07 and the (public review version of) ECSS SMP specification. Further, it outlines how existing SMP2 models can be migrated to ECSS SMP.

INTRODUCTION

Almost 15 years have passed between the first release candidate of the SMP2 specification and the latest draft of the ECSS Working Group towards an ECSS SMP standard. Although major concepts have not been touched, various improvements have been made over the years. On the other hand, major elements of SMP2 (namely Assembly and Schedule files covering a run-time configuration of model instances) have been de-scoped from ECSS SMP. In the following sections, we try to categorise the major changes into different groups.

CONCEPTS REMOVED FROM THE STANDARD

A number of concepts have been removed from the standard, partially to reduce its scope, partially because they provided a level of flexibility (and hence a possible variation of implementations) that was not used when applying SMP2 in practice.

Level 2 support, namely Assembly and Schedule files

SMP2 as well as E-TM-40-07 contain a concept for the assembly of model instances, namely via an XML file called `Assembly` defining model hierarchy, model interfacing and initial model state. This is supported by an additional XML file called `Schedule` defining the initial loading of the scheduler with simulation events. While it was not mandatory to use these concepts (which were classified as an optional “Level 2”), they have been removed in the draft of ECSS SMP. It is hence no longer possible to exchange a complete simulator assembly between different simulation environments via standardised file format.

Platform Independent Component Model

In SMP2 and E-TM-40-07, the Component Model (set of interfaces to use for communication) has been defined in CORBA IDL, with a specific Platform Mapping to the C++ Platform. In the draft of ECSS SMP, the interfaces are defined in C++ 11 only, as it is not expected that a mapping of a platform independent Component Model to further platforms (such as Java) will be done soon. This significantly reduces complexity of the standard reducing the size of the document, and making it easier to understand.

Mandatory versus Optional Services

SMP2 defines a set of five simulation services, from which only four are mandated to be present, and one (the Resolver) is considered optional. In E-TM-40-07 and the draft of ECSS SMP, all simulation services are mandatory, so that the porting of a model from one simulation environment to another simulation environment cannot fail because of a missing optional service. Without making simulation services mandatory, a model cannot safely use them without putting portability at risk. The ability to define additional environment specific simulation services is however still retained.

Concept of Managed Models

SMP2 defines two different types of models: "Simple" models only implementing a "base version" of most of the interfaces, and "managed" models implementing an "extended version" of these interfaces. For that, "unmanaged" and "managed" versions of the interfaces are defined (e.g. `IObject` and `IManagedObject`). It has been observed that all major SMP2 developments are done making use of Code Generators, and such Code Generators always fully auto-generate the extensions of the "managed" versions of these interfaces. Therefore, the number of interfaces has been reduced, and the "managed" extensions formerly put into extended interfaces are now part of the base interfaces. For the same reason, it has been decided to remove `IDynamicSimulator` and merge it with `ISimulator`.

Services versus Models

It has been realised that Services need to be treated identically to Models in most respects, except for the fact that they are singletons, while the Simulator itself follows a very different state machine (including both its states and its state transitions). Therefore, the inheritance of the `ISimulator` interface has been changed, and mechanisms that Models need to share with Services have to be moved from the `IModel` interface into the `IComponent` interface. The only breaking change of this re-organisation is that the existing type `ModelStateKind` had to be renamed to `ComponentStateKind`, as it now applies to Services as well. Figure 1 shows the result of this clean-up process.

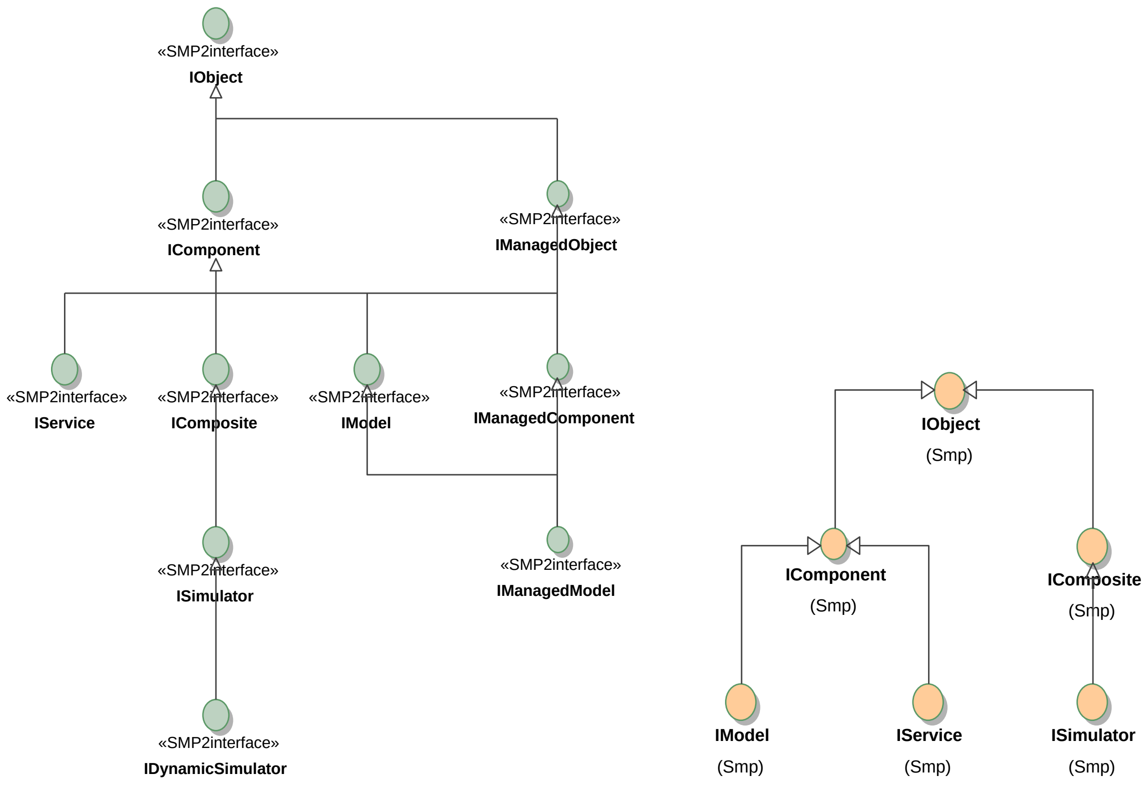


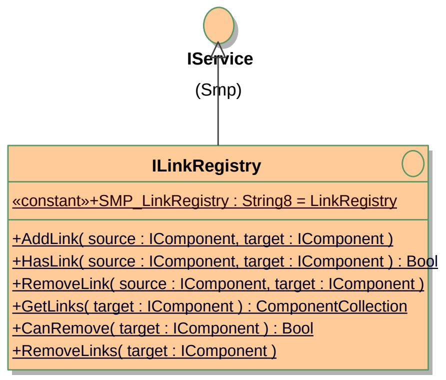
Figure 1: Component Interfaces in SMP2 (left) and ECSS SMP (right)

CONCEPTS ADDED TO THE STANDARD

Configuration file

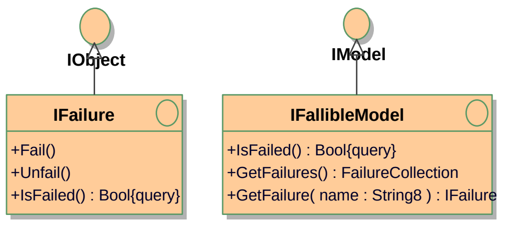
To keep the ability to initialise a simulation from XML files, E-TM-40-07 introduced a new `Configuration` file format. It can be used to set initial field values, as well as loading field values any time later during a simulation.

Link Registry Service


ISservice
(Smp)
ILinkRegistry
«constant»+SMP_LinkRegistry : String8 = LinkRegistry
+AddLink(source : IComponent, target : IComponent)
+HasLink(source : IComponent, target : IComponent) : Bool
+RemoveLink(source : IComponent, target : IComponent)
+GetLinks(target : IComponent) : ComponentCollection
+CanRemove(target : IComponent) : Bool
+RemoveLinks(target : IComponent)

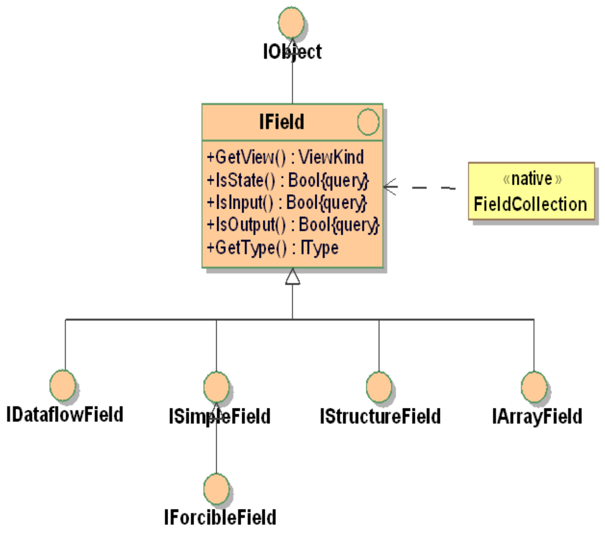
In E-TM-40-07, mechanisms were added to support dynamic referencing of components by other components. The mechanisms introduced in SMP2 (`IReference` and `IManagedReference`) have been merged into `IReference`, and extended by a new operation `RemoveComponent()` to remove a reference at run-time. To maintain all existing references within a simulation, a Link Registry has been introduced as a new Service. Finally, components supporting this have to implement `ILinkingComponent`.

Failures


IObject
IModel
IFailure
+Fail()
+Unfail()
+IsFailed() : Bool{query}
IFallibleModel
+IsFailed() : Bool{query}
+GetFailures() : FailureCollection
+GetFailure(name : String8) : IFailure

For models providing one or several failures, two new interfaces have been added in E-TM-04-07, namely `IFailure` and `IFallibleModel`. A Model can have any number of failures. It is considered failed if at least one of its failures is activated.

Field Interfaces, including Forcing of Fields



While SMP2 only provides a publication mechanism of fields via their type and memory address, E-TM-40-07 and the draft of ECSS SMP add the ability to publish fields as objects implementing the new `IField` interface.
To support all types of fields (simple types, structures and arrays), three dedicated interfaces are derived from `IField`. Further, each field may implement the additional interface `IDataflowField` so that it can participate in direct inter-model data flow. Transfer of data values from source to target fields can then be triggered directly by the participating models, not only by the scheduler.
For fields of simple types, the interface `IForcibleField` adds the ability to force the value of the field, and to unforce it again. This is often used in combination with model failures, as introduced above.

Return Parameters

In SMP2, the return type of an operation is specified as an additional parameter to `PublishOperation()`, or as a link to a type (in the Catalogue). In E-TM-40-07 and the draft of ECSS SMP, an additional parameter with the new `ParameterDirectionKind = PDK_Return` has to be published (in C++) or added to the Operation (in the Catalogue). This is consistent with the UML meta-model, and allows defining a description and additional meta-data (attributes) for the return value of an operation.

Visibility Kinds

In the publication of Fields, the existing Boolean for the view state has been replaced by an enumeration `ViewKind` offering four different visibility levels (None, Debug, Expert and All). In addition, this concept has been extended to Operations and Properties. While his has no immediate effect on running a simulation, such meta-data can be used by user interfaces to provide context sensitive filtering of the level of information to provide to the end-users.

String Support

The `AnySimple` type has been extended to support character strings, e.g. in data flow simulations.

Automatic Data Flow Propagation

SMP2 and E-TM-40-07 support a data flow mechanism where changes to the value of a model output field can be propagated to the value of a model input field. The transfer of values between input and output fields was performed through an explicit task that is scheduled externally to a model. ECSS SMP extends this by allowing the automatic propagation of the data transfer to an input field when a model updates an output field.

Improved Support for integration of Emulators

The use of SMP2 in various simulation environments has highlighted the need to provide additional features in the Scheduler, Time Keeper and Event Manager services to enable an integration of a Processor Emulator without any environment specific extensions. The Time Keeper interface has been extended by a new operation to propagate simulation time forward (driven by a Processor Emulator), and the Scheduler interface provides two new operations to query for the event currently executed, and of the time of the next event on the schedule. Finally, the Event Manager has been extended by two additional events emitted before and after changes of the simulation time.

CLEAN-UP DUE TO THE USE OF C++ 11

Removed the need for tailoring in `Platform.h`

Due to the use of C++ 11, it has been possible to fully base the ECSS SMP Standard on the C++ standard, without the need for a platform specific tailoring (done in `Platform.h` in the SMP2 specification). As a consequence, it is no longer necessary (or allowed) to provide a mapping of the Primitive Types to C++ types (as C++ 11 defines all necessary types, including signed and unsigned 64 bit integer types).

Use of new features introduces with C++ 11

All dynamic exception specifications (`throw` statements in declarations) have been removed, as they are deprecated. The `noexcept` specification has been used in specific cases to ensure a valid runtime behaviour (e.g. in destructor declarations).

The UUID class has been modified to support constant expression definitions. Such constant expressions have been introduced to define UUIDs and Strings in header files. This provides well-defined linkage behaviour compared to the `static` constants used in SMP2.

Move semantic has been introduced to `AnySimple`. This has been primarily to support efficient transfer of strings (which have been added, see above).

The declaration for a standard UUID hash function has been added. This allows using UUIDs in unordered containers. It can help to significantly ease an implementation of an efficient Type Registry with a look-up mechanism by UUID.

Replacement of std::vector by C++ collection

Various interfaces have an operation returning a collection of an interface type. In SMP2 and E-TM-40-07, this has been defined via `std::vector`, restricting the implementation to a specific C++ collection type. In the draft of ECSS SMP, this has been replaced by a new interface `ICollection` which can be implemented against `std::vector`, but as well against other C++ collection types.

Removal of all implementation code

No source code is included in the draft of ECSS SMP, only declarations. This holds even for exceptions.

BREAKING CHANGES OF EXISTING INTERFACES

Specification UUID versus Implementation UUID

In SMP2, a model defined in a Catalogue (with a specification UUID) can have different implementations (in the same package, or in different packages), all providing a different implementation UUID. As this concept has not been used, and is not even supported by many SMP2 development environments (e.g. UMF), it has been removed. This simplifies the Package file format and the `IFactory` and `ISimulator` interfaces.

Removed Setters for Name, Description and Parent

The hierarchy of objects in a simulation is expected to be fixed, and defined during creation. Therefore, the operations `SetName()`, `SetDescription()` and `SetParent()` have been removed, together with the interfaces `IManagedObject` and `IManagedComponent`. Name, description and parent are typically set in a constructor, but cannot be changed externally via standardised interfaces anymore. This avoids that an object or component in a simulation can e.g. rename other objects in the simulation, or change the hierarchy of components.

Renamed Operations to Query for global Event Identifier or Log Message Type

The two operations `GetEventId()` and `GetLogMessageKind()` of the corresponding services have been renamed to `QueryEventId()` and `QueryLogMessageKind()`, to express that they either return an existing identifier, or create a new one.

Defined each Type in a dedicated header file

For consistency, and to ease implementation of code generators, each type defined in ECSS SMP has been put into a dedicated header file with the name of the type. This includes exceptions, which are no longer defined within interfaces.

NON-BREAKING CHANGES OF EXISTING INTERFACES

Removed limitation of names to 32 characters

While SMP2 limits names of objects to 32 characters, this limit has been removed in ECSS SMP.

Extended Type Definition mechanism for Integer Types

User-defined integer types are no longer restricted to a subset of the available integer types, i.e. it is now possible to define an integer type based on UInt64. Further, user-defined integer types can define an optional unit (a String).

Extended Persistence from Components to Objects

In SMP2, the mechanism of self-persistence is defined as an extension of `IComponent`, hence limited to components. In ECSS SMP, it has been derived from `IObject`, making it available to all objects in a simulation. Therefore, self-persistence can e.g. be used by fields or failures (both `IField` and `IFailure` have been introduced by ECSS SMP).

Added operations to existing interfaces

Some interfaces have been extended by additional operations, to support additional use cases that had to be done outside of the SMP2 specification before.

`IComponent::GetUuid()`	added (query a component for its UUID defined in a Catalogue)
`IScheduler::GetCurrentEventId()`	added (Improved Support for Integration of Emulators)
`IScheduler::GetNextScheduledEventTime()`	added (Improved Support for Integration of Emulators)
`ITimeKeeper::SetSimulationTime()`	added (Improved Support for Integration of Emulators)
`IEventManager::PreSimTimeChange` event	added (Improved Support for Integration of Emulators)
`IEventManager::PostSimTimeChange` event	added (Improved Support for Integration of Emulators)
`IStorageWriter::GetStateVectorFileName()`	added (Allows writing your own state vector file)
`IStorageWriter::GetStateVectorFilePath()`	added (Allows writing your own state vector file)
`IStorageReader::GetStateVectorFileName()`	added (Allows reading your own state vector file)
`IStorageReader::GetStateVectorFilePath()`	added (Allows reading your own state vector file)

Moved operations between existing interfaces

Moved `GetParent()` operation from `IComponent` to `IObject`, so that each object in a simulation exposes its parent.

Moved `IModel` operations to `IComponent`, so that they are shared with `IService` (see above).

Merged `IDynamicSimulator` into `ISimulator`: The remaining operations of `IDynamicSimulator` have been moved to `ISimulator`, and `IDynamicSimulator` has been removed.

MIGRATION FROM SMP2 TO ECSS SMP

This section provides an assessment of the work to migrate existing SMP2 models to the current draft of the ECSS SMP Standard. It has to be noted that this is an initial assessment, as the final ECSS SMP Standard may differ from the draft available at the time of the preparation of this paper. However, the majority of changes can be considered stable.

Many of the changes are in source code that is typically auto-generated, so the migration to a toolset supporting ECSS SMP where the auto-generated code is re-generated is in many cases sufficient. In few cases, manual changes of source code will be required. In this section, we summarise the changes that may break existing source code. Neither new operations added to existing interfaces nor new interfaces are included in this list, as they both do not cause existing code to break. However, for a full migration to ECSS SMP, it is clearly recommended to make use of these new features where applicable.

Modifications to existing operations of existing interfaces

The lists below include all interfaces of SMP2. Many of them did not have changes applied. For those that have changes, the changes are often backwards compatible, as an operation has been moved into a base interface. Interface changes that are likely to break existing code (highlighted in red) are in the area of Managed Interfaces, Dynamic Simulator and the use of Factories, and in `IPublication` (mainly used in auto-generated code).

Table 1: SMP2 Interfaces in `namespace Smp`

Interface	Operation	Change
`IAggregate`		None
`IComponent`	`GetParent`	Moved to `IObject`.
`IComposite`		None
`IContainer`		None
`IDynamicInvocation`		None
`IDynamicSimulator`	Interface has been deleted, and content has been merged with `ISimulator`.	
	`RegisterFactory`	Moved to `ISimulator`.
	`CreateInstance`	Moved to `ISimulator`.
	`GetFactory`	Moved to `ISimulator`.
	`GetFactories`	Removed
`IEntryPoint`	`GetOwner`	Replaced by `IObject::GetParent`.
`IEventSink`		None
`IEventSource`		None
`IFactory`	`GetSpecification`	Renamed to `GetUuid`.
	`GetImplementation`	Removed
`IModel`	`GetState`	Moved to `IComponent`. Changed return type to `ComponentStateKind`.
	`Publish`	Moved to `IComponent`.
	`Configure`	Moved to `IComponent`.
	`Connect`	Moved to `IComponent`.
`IObject`		None
`IPersist`		None
`IPublication`	`PublishField`	Changed `view` parameter from `Bool` to `ViewKind`.
	`PublishArray`	Changed `view` parameter from `Bool` to `ViewKind`. Changed `type` parameter from `SimpleTypeKind` to `PrimitiveTypeKind`
	`PublishOperation`	Removed `returnTypeUuid` parameter. Added `view` parameter of type `ViewKind`.
	`PublishProperty`	Added `view` parameter of type `ViewKind`.
	Replaced `Get<...>Value` operations by new operation `GetField`, and by operations provided by new interfaces `ISimpleField` and `IArrayField`.	
	`GetFieldValue`	Replaced by `ISimpleField->GetValue()`
	`SetFieldValue`	Replaced by `ISimpleField->SetValue()`
	`GetArrayValue`	Replaced by `IArrayField->GetValues()`
	`SetArrayValue`	Replaced by `IArrayField->SetValues()`
`IReference`		None
`IRequest`		None
`IService`		None
`ISimulator`		No functions changed, by base interface changed to `IObject`

Table 2: SMP2 Interfaces in the `namespace Smp::Management`

Interface	Operation	Change
`IEntryPointPublisher`	Interface moved into `namespace Smp`.	
`IEventConsumer`	Interface moved into `namespace Smp`.	
`IEventProvider`	Interface moved into `namespace Smp`.	
`IManagedComponent`	`SetParent`	Operation and interface removed. This has to be set during construction.
`IManagedContainer`	Interface content has been moved to `IContainer`, and interface has been removed.	
`IManagedModel`	Interface has been removed. Replaced `Get<...>Value` operations by new operation `IComponent::GetField()`, and by operations provided by new interfaces `ISimpleField` and `IArrayField`:	
	`GetFieldValue`	Replaced by `ISimpleField->GetValue()`
	`SetFieldValue`	Replaced by `ISimpleField->SetValue()`
	`GetArrayValue`	Replaced by `IArrayField->GetValues()`
	`SetArrayValue`	Replaced by `IArrayField->SetValues()`
`IManagedObject`	`SetName`	Operations and interface removed. This has to be set during construction.
	`SetDescription`	
`IManagedReference`	Interface content has been moved to `IReference`, and interface has been removed.	

Table 3: SMP2 Interfaces in the `namespace Smp::Publication`

Interface	Operation	Change
`IClassType`		None
`IEnumerationType`		None
`IPublishOperation`	`PublishParameter`	Added `direction` with default `PDK_In`.
`IStructureType`	`AddField`	Changed `view` parameter from `Bool` to `ViewKind`.
`IType`	`GetSimpleType`	Renamed to `GetPrimitiveType`. Changed return type to `PrimitiveTypeKind`.
	`Publish`	Changed `view` parameter from `Bool` to `ViewKind`.
`ITypeRegistry`	`GetType`	Changed `type` parameter to `PrimitiveTypeKind`.
	`AddFloatType`	Changed `type` parameter to `PrimitiveTypeKind`.
	`AddIntegerType`	Added `unit` parameter of type String. Changed `type` parameter to `PrimitiveTypeKind`.

Table 4: SMP2 Interfaces in the `namespace Smp::Services`

Interface	Operation	Change
`IEventManager`	`GetEventId`	Renamed to `QueryEventId`.
	`Emit`	Added optional argument `synchronous` which defaults to `true`.
`ILogger`	`GetLogMessageKind`	Renamed to `QueryLogMessageKind`.
`IResolver`		None
`IScheduler`	`AddImmediateEvent`	Changed from `void` to returning and `EventId`.
	`Add<...>TimeEvent`	Renamed parameter `count` to `repeat`. Semantic kept.
`ITimeKeeper`		None

Renamed Types due to the clean-up

Due to the clean-up and re-organisation of types and interfaces, a few types had to be renamed.

Old Type Name	New Type Name	Justification of the change
`SimpleTypeKind`	`PrimitiveTypeKind`	The enumeration actually only contains the primitive types of the type system. The term “simple type” still exists, but includes types (Float, Integer and Enumeration) that can be mapped to a primitive type.
`ModelStateKind`	`ComponentStateKind`	Models and services now share a common state machine.

Header file reorganization

Due to the reorganisation of header files and especially exceptions, several exceptions have changed their location, both for the header file to include, and for the namespace to use when throwing the exception.

REFERENCES

[1] SMP2 Standard: SMP2 Handbook (EGOS-SIM-GEN-TN-0099), SMP2 Metamodel (EGOS-SIM-GEN-TN-0100), SMP2 Component Model (EGOS-SIM-GEN-TN-0101) and SMP 2.0 C++ Mapping (EGOS-SIM-GEN-TN-0102), all Issue 1 Revision 2 from 28 October 2005

[2] ECSS-E-TM-40-07: Volume 1A (Principles and requirements), Volume 2A (Metamodel), Volume 3A (Component model), Volume 4A (C++ Mapping) and Volume 5A (SMP usage), all from 25 January 2011

[3] Draft of ECSS SMP, internal version for review, provided to the ECSS SMP Working Group on 31/12/2016