Simulation Exploration Experience (SEE)

The "SEE HLA Starter Kit"

Table of contents

1 Introduction

This document contains an overview of the "SEE HLA Starter Kit", a software framework for ease the development of HLA Federate in the context of the "Simulation Exploration Experience" (SEE) project; in particular: Section 2 describes the "SEE HLA Starter Kit" objectives; Section 3 highlights the main "services" that the Starter Kit is going to provide; Section 4 reports the high-level Architecture of the SEE HLA Starter Kit Framework (SKF). In the end (Appendix C), the Java code related to a SEE Federate developed by using the Kit is reported.

2 SEE HLA Starter Kit Main Objectives and Development Method

The SEE HLA Starter Kit aims to ease the development of HLA federates in the context of the Simulation Exploration Experience (SEE) project by providing a software framework with related documentation, user guide and reference examples. Specifically, the SEE HLA Starter Kit will allow developers to focus on the specific aspects of their own HLA federates rather than dealing with the common HLA issues such as the management of the simulation time, the connection on the RTI, etc. Moreover, the SEE HLA Starter Kit will support the implementation of SEE Dummy and Tester Federate so to allow a more accurate and effective testing. These features should improve the reliability of SEE Federates and thus reduce the problems arising during the final integration and testing phases of the SEE project.

The SEE HLA Starter Kit will be developed according to the concept of Object HLA; this approach to the development of SEE Federate could benefit also from the "Object HLA" features and functionalities provided by the "Pitch Developer Studio" or similar IDE.

The design and implementation of the SEE HLA Starter Kit will be centered on typical Software Engineering methods and, in particular, on an Agile software development process: after a high-level planning and analysis necessary to outline the scope of the project, a series of weekly iterations, that will include analysis, design, development and testing activities, will be performed (see Figure 1).

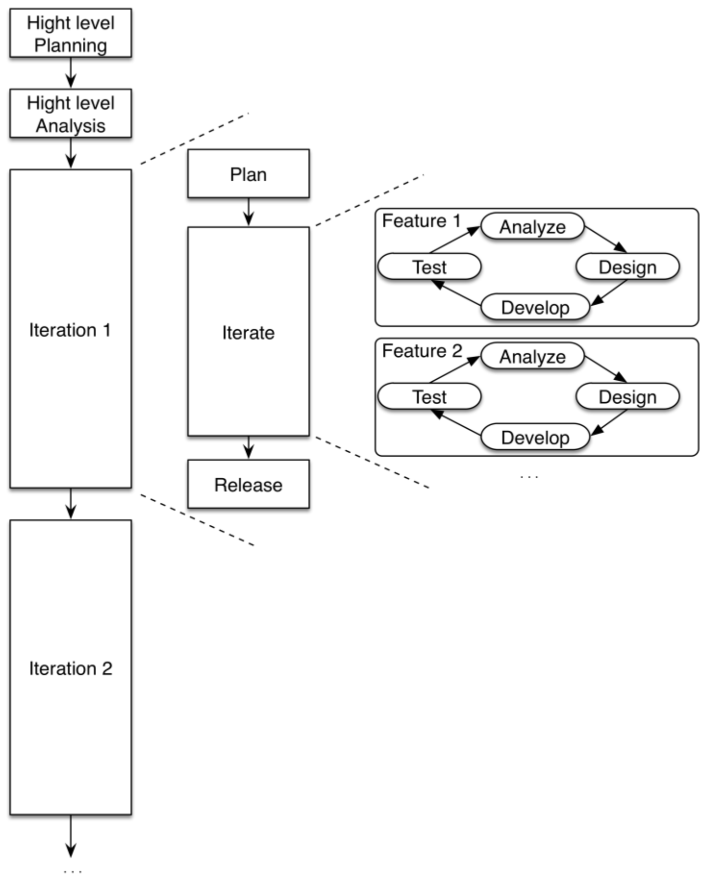

Figure 1: The exploited Agile software development process.

3 Starter Kit Main Services

For the definition of the "SEE HLA Starter Kit" and related development framework (SKF), some important "services" to be provided are under identification. In the following, a starting list is reported by organizing them into categories and by also specifying the related RTI service type as defined by the HLA standard.

CONNECTION MANAGEMENT services *(RTI-Coordination Services)*:

- management of the connection parameters of the VPN;
- management of the connection of the federates to a SEE Federation execution;
- management of the resign of the federates from a SEE Federation execution;
- ...;

FOM MANAGEMENT services *(RTI-Information Services)*:

- FOM module publication services;
- ...;

INTERACTION MANAGEMENT services *(RTI-Information Services)*:

- publishing services;
- subscribing services;
- ....;

SIMULATION TIME MANAGEMENT services *(RTI-Synchronization Services)*:

- simulation time handling;
- time standard conversions;
- ...;

COORDINATES FRAME MANAGEMENT services:

- transformations among SEE Coordinate Systems;
- publication and subscription of SEE Reference Frames;
- ...;

OWNERSHIP TRANSFER services:

- ...;

DATA DISTRIBUTION MANAGEMENT services

- ...;

LOGGING services:

- management of SEE-specific logs;
- ...;

TESTING services:

- IP Configuration testing;

- MS Windows Firewall testing ;
- LRC/CRC parameters testing;
- ...;

4 Starter Kit Framework (SKF) Architecture

The SEE HLA Starter Kit will provide the SEE teams with a framework (SKF) consisting of a set of Java classes that are able to handle the main and common mechanisms of a SEE HLA Federate and that can be easily extended and integrated with federate-specific code. As a consequence, a SEE Federate will be composed by a set of classes provided by the SKF that are extended and/or integrated with classes developed by the teams for implementing the specific logic and "services" of their federates; in particular, by using an "Object HLA" approach, the SKF provides the following main services (see Section 3 for a more complete list):

1. *Simulation Time Management:* management of the *"logical time"* of a SEE Federate during the simulation execution;
2. *Connection Management:* handling of the phases of set-up, hold-up and close-up of the connection to the RTI (Run Time Infrastructure);
3. *Interaction Management:* handling of the notification coming from/direct to the RTI concerning the interactions with other federates.

4.1 SKF packages

The SEE HLA Starter Kit sources are organized into a hierarchy of Java packages and sub-packages, where each package contains a set of classes and interfaces that implement specific functionalities; the main packages are:

- **skf.core**, implements the kernel of the SFK framework. It includes the fundamental *skf.core.SEEAbstractFederate* and *skf.core. SEEAbstractFederateAmbassador* classes that provide the basic functionalities to manage a SEE Federate. This package also includes a sub-packages that implementing a specific core-level service; this is:
  - **skf.core.observer,** which manages the updates concerning both *Object Attributes* and *Interactions*; this package allows a SEE

Federate to be notified whenever its *FederateAmbassador* receives an attribute/interaction update.

- **skf.config**, which contains the collection of classes that manage the configuration parameters provided by a "*.json*" file.
- **skf.utility**, which contains several miscellaneous utility classes, such as transformations among SEE Coordinate Systems, time standard conversions and Windows Firewall Check.
- **skf.logging**, which contains a set of classes used to track down any problems or error occurred during the execution of the SEE Federate; these information are stored into a "*skf.log*" file.
- **skf.exception**, which contains some definitions of exceptions that are used for handling dysfunctional events throughout the SKF framework.
- **skf.model,** which contains some classes to facilitate publishing, subscribing and the data updating of both an ObjectClass and an InteractionClass. This package also includes the two following sub-packages that implement specific services:
  - **skf.model.object,** which defines the annotations that have to be used by the programmer, so to create an 'ObjectClass' instance compatible with the kit;
  - **skf.model.interaction,** which defines the annotations that have to be used by the programmer, so to create an 'InteractionClass' compatible with the kit;
  - **skf.model.parser,** the parser of both the 'ObjectClass' and the 'InteractionClass''.
- **skf.coder,** which contains a set of classes used to coding and decoding an objectModel.

Figure 2 shows the (current) UML Package Diagram of the SKF framework.

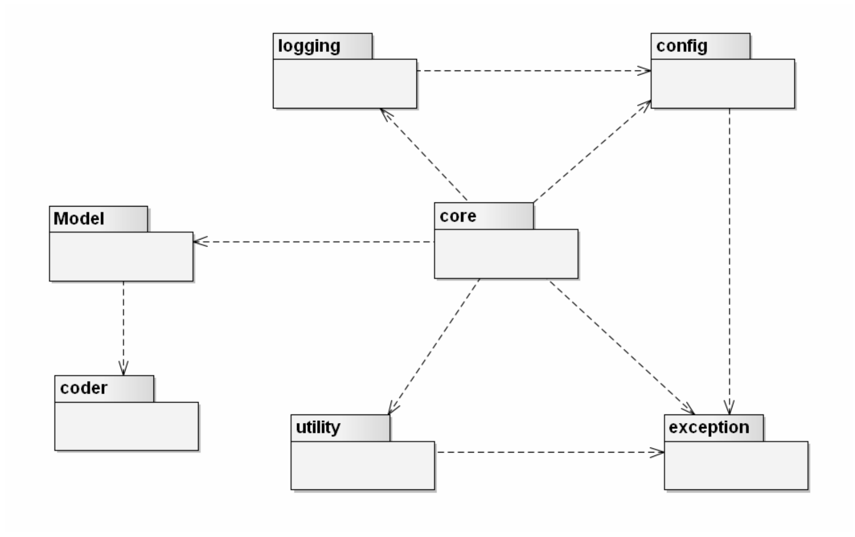

Figure 2: SEE HLA Starter Kit, UML Package Diagram.

4.1.1 The skf.core package

The architecture of the skf.core package is shown in Figure 3 by using a UML Class Diagram; in the following its main classes are briefly described.

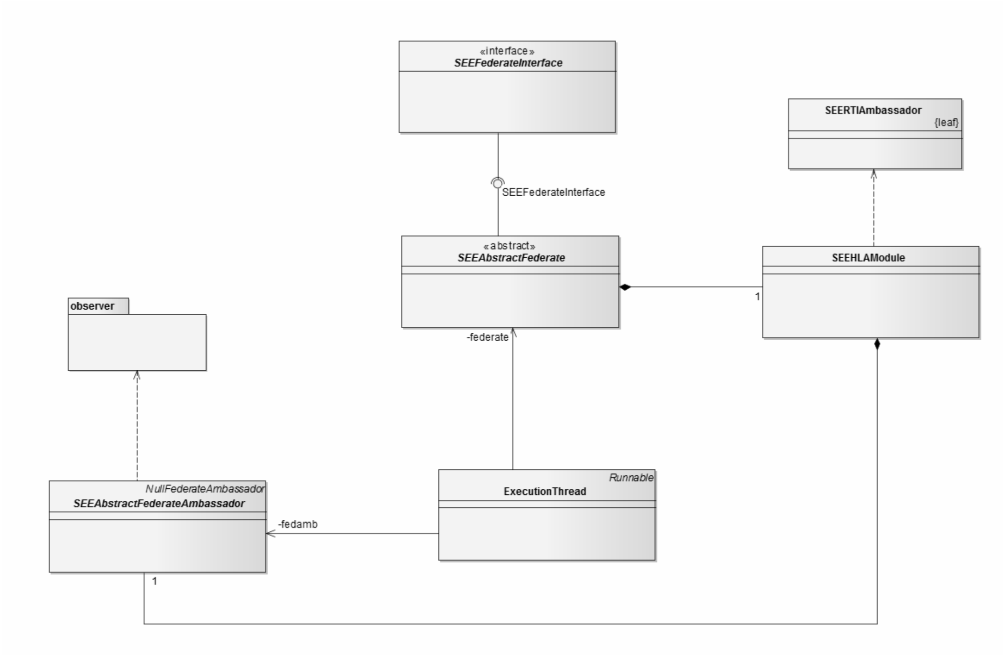

Figure 3 : The UML class diagram of the skf.core package.

The *SEEAbstractFederate* class manages the life cycle of a SEE Federate; it provides functionalities to configure and connect/disconnect the SEE federate to/from the SEE Federation. In particular, the *SEEAbstractFederate* class provides a concrete SEE Federate with the management of its life cycle (FLCM), as a consequence, a SEE working team has only to define the specific behavior of its SEE Federate (through the "processing and update data" state, see Figure 4). Thanks to the FLCM a SEE working team can spend more time to improve its SEE Federate without worrying about low-level implementation details since the SKF manages them.

The *ExecutionThread* class handles the execution of a SEE Federate in the simulation environment.

The *SEEAbstractFederateAmbassador* class implements the methods that are called by the RTI for interacting with the Federate (RTI callback methods); along with the RTI Ambassador interface, which is used by the Federate to access the RTI services.

With reference to the life cycle of a SEE Federate provided by the *SEEAbstractFederate*, in Figure 4 it is shown through a UML statechart diagram. Specifically, in the *load configuration* state, the SKF loads the configuration parameters from a *.json* file. A transition to the *startup* state happens if the configuration parameters are valid and during the state transition a connection to the SEE Federation execution platform (HLA RTI) is performed. If the configuration parameters are invalid a state transition to the *shutdown* state is performed. In this latter state, all the resources engaged by the SKF framework are de-allocated and the lifecycle terminates. In the *startup* state, the SKF checks the connection status. If the connection is not established the lifecycle ends with a transition to the *shutdown* state. Otherwise, a transition to the *initialization* state is performed; in this state, the SEE Federate could perform additional operation for exchanging initialization objects before entering the "running" state (and thus the time advancement loop: waiting for TAG -> processing and update data -> make TAR request), as an example, the

Federate could publish and subscribe some SEE information (e.g. Reference Frames, Interactions, etc.). After that, the SKF activates the time management thread and a transition to the *running* state is performed. The *running* state handles the simulation execution through three states: (i) *Waiting for TAG*: the SKF waits for the "TAG (Time Advance Grant) Callback" from the RTI. When the callback is received a transition to the *processing and update data* state is performed; (ii) *processing and update data*: the "logical time" is updated, the behavior of the specific SEE Federate defined by the SEE working team are executed, and then a transition to the *make TAR request* state is performed; (iii) *make TAR (Time Advance Request) request*: the SKF requests to the RTI the grant for the next "logical time".

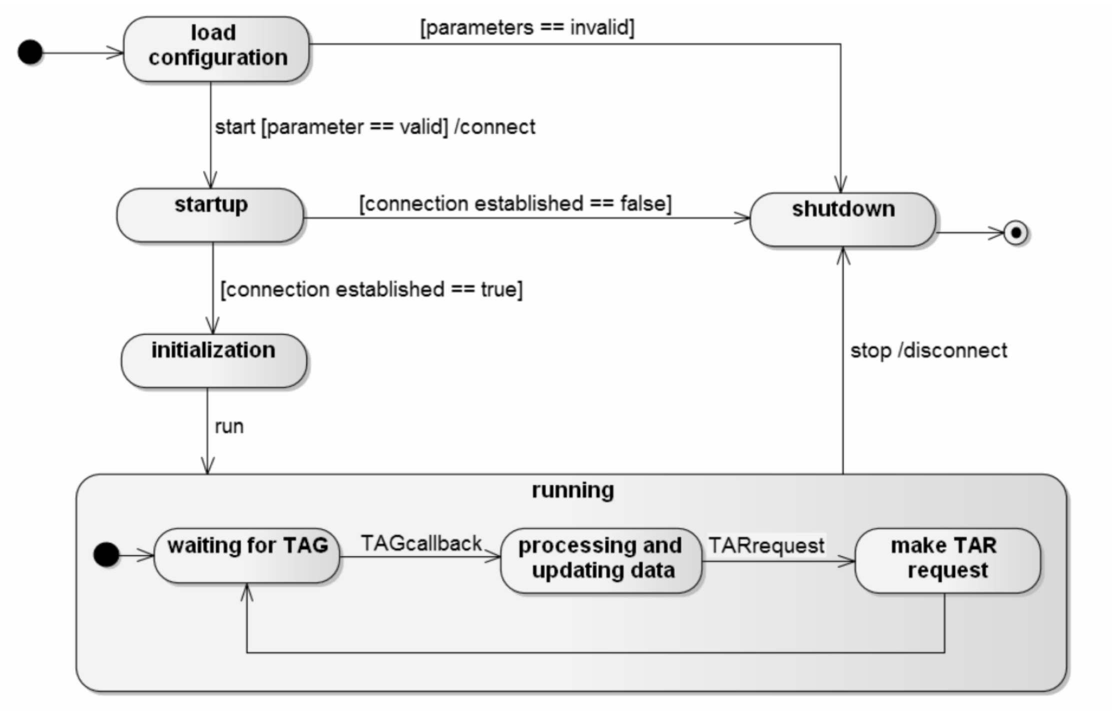

Figure 4: a UML statechart of the life cycle.

4.1.1.1 The skf.core.observer package

This package implements the *Observer Pattern*, in which an object classed *Subject*, maintains a list of its dependents, called *Observers*, and notifies them automatically of any state changes.

The above-mentioned pattern has been exploited to the specific problem related to the notification of a SEE Federate about the updates of the subscribed *ObjectClasses and/or Interactions*. In this case, the *Subject* class represents, according to the patter notation, the subject component; whereas the implementation of the concrete *Observer* class is in charge of the developer.

The *Subject* class notifies its *Observers* whenever a SEE Federate updates a subscribed object and/or interaction.

The architecture of the above described implementation of the skf.config.observer package is shown in Figure 5 by using a UML Class Diagram.

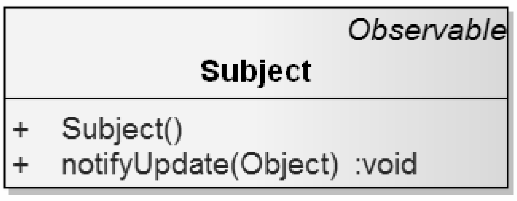

Figure 5: The UML class diagram of the skf.config.observer package.

4.1.2 The skf.config package

The architecture of the skf.config package is shown in Figure 6 by using a UML Class Diagram.

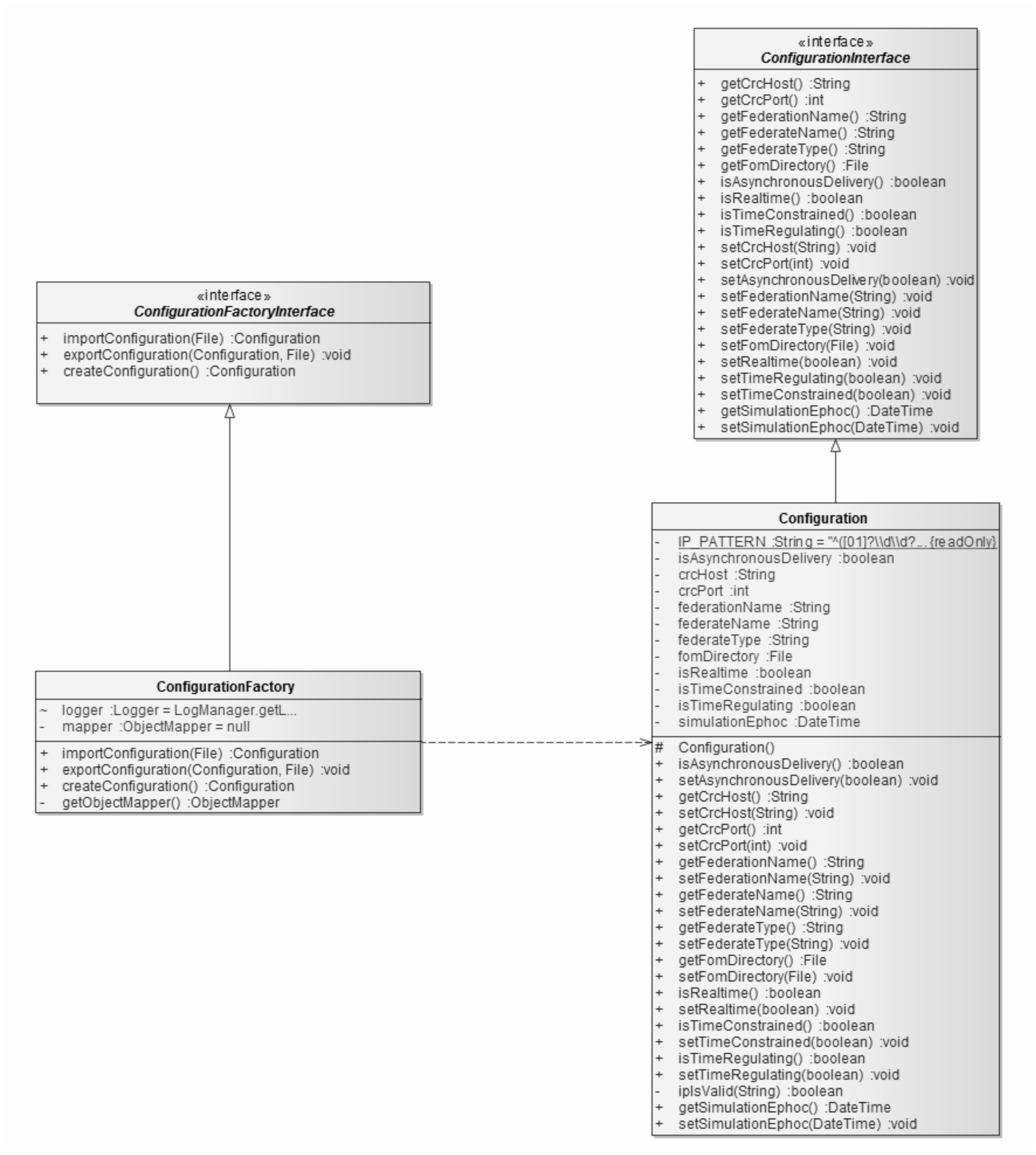
«interface»
ConfigurationInterface
+ getCrcHost() :String
+ getCrcPort() :int
+ getFederationName() :String
+ getFederateName() :String
+ getFederateType() :String
+ getFomDirectory() :File
+ isAsynchronousDelivery() :boolean
+ isRealtime() :boolean
+ isTimeConstrained() :boolean
+ isTimeRegulating() :boolean
+ setCrcHost(String) :void
+ setCrcPort(int) :void
+ setAsynchronousDelivery(boolean) :void
+ setFederationName(String) :void
+ setFederateName(String) :void
+ setFederateType(String) :void
+ setFomDirectory(File) :void
+ setRealtime(boolean) :void
+ setTimeRegulating(boolean) :void
+ setTimeConstrained(boolean) :void
+ getSimulationEphoc() :DateTime
+ setSimulationEphoc(DateTime) :void
«interface»
ConfigurationFactoryInterface
+ importConfiguration(File) :Configuration
+ exportConfiguration(Configuration, File) :void
+ createConfiguration() :Configuration
Configuration
- IP_PATTERN :String = "^([01]?\\d\\d?... {readOnly}
- isAsynchronousDelivery :boolean
- crcHost :String
- crcPort :int
- federationName :String
- federateName :String
- federateType :String
- fomDirectory :File
- isRealtime :boolean
- isTimeConstrained :boolean
- isTimeRegulating :boolean
- simulationEphoc :DateTime
# Configuration()
+ isAsynchronousDelivery() :boolean
+ setAsynchronousDelivery(boolean) :void
+ getCrcHost() :String
+ setCrcHost(String) :void
+ getCrcPort() :int
+ setCrcPort(int) :void
+ getFederationName() :String
+ setFederationName(String) :void
+ getFederateName() :String
+ setFederateName(String) :void
+ getFederateType() :String
+ setFederateType(String) :void
+ getFomDirectory() :File
+ setFomDirectory(File) :void
+ isRealtime() :boolean
+ setRealtime(boolean) :void
+ isTimeConstrained() :boolean
+ setTimeConstrained(boolean) :void
+ isTimeRegulating() :boolean
+ setTimeRegulating(boolean) :void
- ipIsValid(String) :boolean
+ getSimulationEphoc() :DateTime
+ setSimulationEphoc(DateTime) :void
ConfigurationFactory
~ logger :Logger = LogManager.getL...
- mapper :ObjectMapper = null
+ importConfiguration(File) :Configuration
+ exportConfiguration(Configuration, File) :void
+ createConfiguration() :Configuration
- getObjectMapper() :ObjectMapper

Figure 6: The UML class diagram of the skf.config package.

This package manages the configuration parameters of the SKF framework, which are stored in a *.json* file. The parameters concern the SEE federation aspects, such as the name and the type of the SEE federation (“federationName” and “federationType”), and the FOM directory (“fomDirectory”).

The *Configuration* class manages the configuration parameters. The *ConfigurationFactory* class implements the *ConfigurationFactoryInterface* and manages the creation, loading and storing of a Configuration object.

4.1.3 The skf.logging package

The architecture of the skf.logging package is shown in Figure 7 by using a UML Class Diagram.

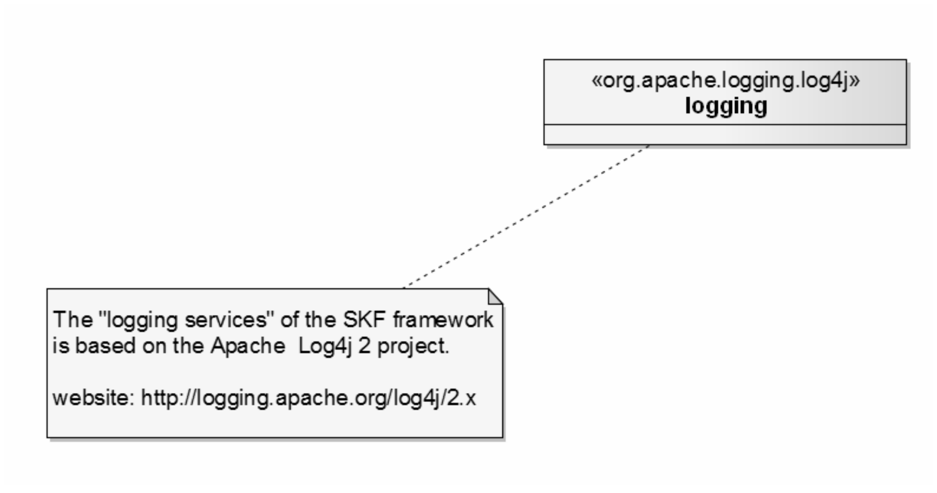
«org.apache.logging.log4j»
logging
The "logging services" of the SKF framework is based on the Apache Log4j 2 project.
website: http://logging.apache.org/log4j/2.x

Figure 7: The UML class diagram of the skf.logging package.

The SKF framework uses the *Apache Log4j 2* library to manage the log events generated by the framework during the simulation execution.

4.1.4 The skf.exception package

The architecture of the skf.exception package is shown in Figure 8 by using a UML Class Diagram.

This package contains classes used to handle the unexpected situations that might occur during the simulation execution.

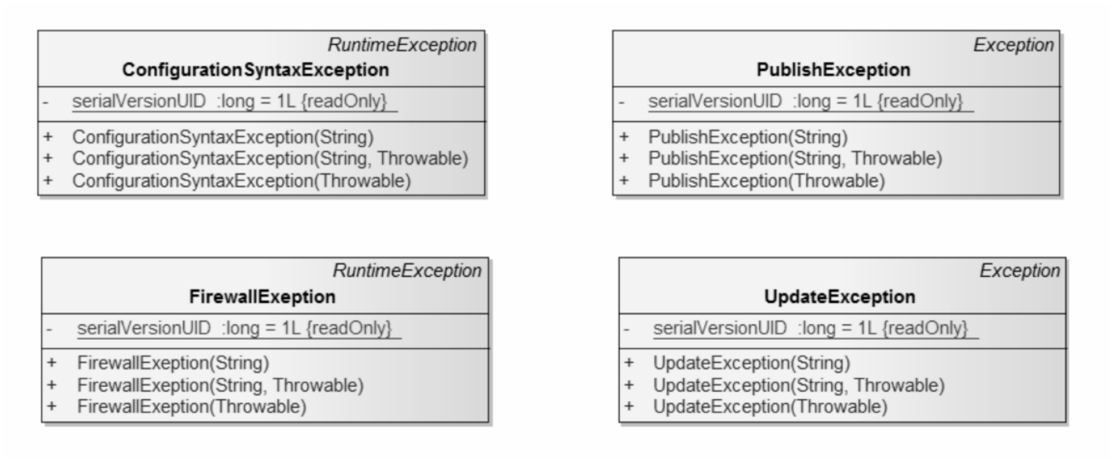
RuntimeException
ConfigurationSyntaxException
- serialVersionUID :long = 1L {readOnly}
+ ConfigurationSyntaxException(String)
+ ConfigurationSyntaxException(String, Throwable)
+ ConfigurationSyntaxException(Throwable)
Exception
PublishException
- serialVersionUID :long = 1L {readOnly}
+ PublishException(String)
+ PublishException(String, Throwable)
+ PublishException(Throwable)
RuntimeException
FirewallExeption
- serialVersionUID :long = 1L {readOnly}
+ FirewallExeption(String)
+ FirewallExeption(String, Throwable)
+ FirewallExeption(Throwable)
Exception
UpdateException
- serialVersionUID :long = 1L {readOnly}
+ UpdateException(String)
+ UpdateException(String, Throwable)
+ UpdateException(Throwable)

Figure 8: The UML class diagram of the skf.exception package.

4.1.5 The skf.coder package

The architecture of the skf.coder package is shown in Figure 9 by using a UML Class Diagram.

This package contains classes used to encode and decode the fields of an *ObjectModel* during both the phases of publication and updating.

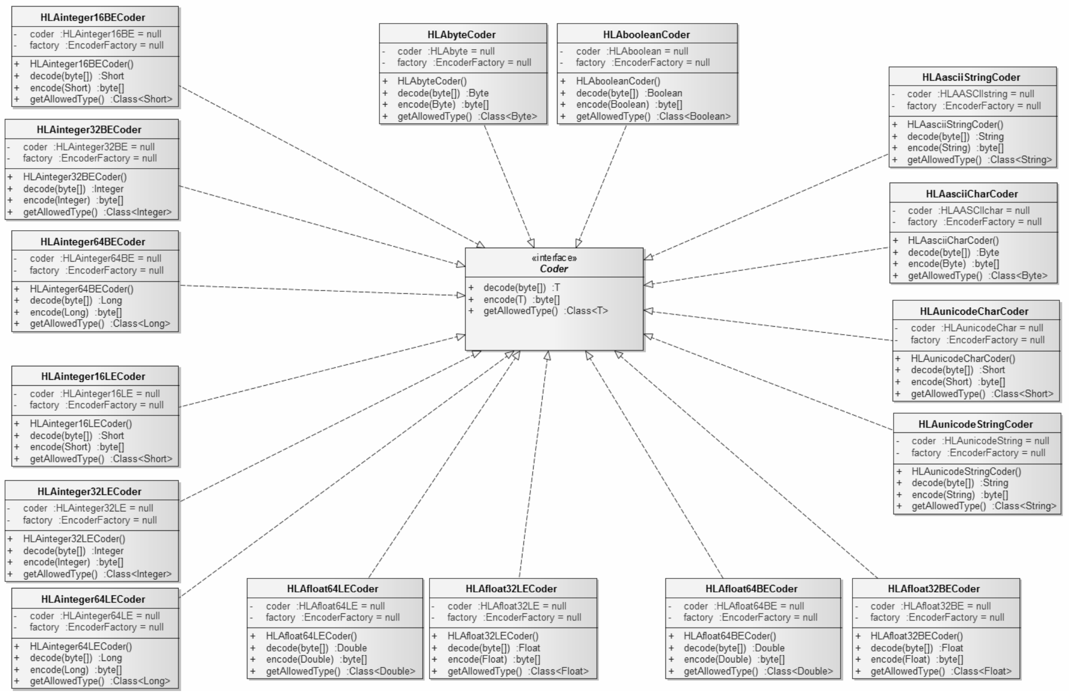
HLAinteger16BECoder
- coder :HLAinteger16BE = null
- factory :EncoderFactory = null
+ HLAinteger16BECoder()
+ decode(byte[]) :Short
+ encode(Short) :byte[]
+ getAllowedType() :Class<Short>
HLAinteger32BECoder
- coder :HLAinteger32BE = null
- factory :EncoderFactory = null
+ HLAinteger32BECoder()
+ decode(byte[]) :Integer
+ encode(Integer) :byte[]
+ getAllowedType() :Class<Integer>
HLAinteger64BECoder
- coder :HLAinteger64BE = null
- factory :EncoderFactory = null
+ HLAinteger64BECoder()
+ decode(byte[]) :Long
+ encode(Long) :byte[]
+ getAllowedType() :Class<Long>
HLAinteger16LECoder
- coder :HLAinteger16LE = null
- factory :EncoderFactory = null
+ HLAinteger16LECoder()
+ decode(byte[]) :Short
+ encode(Short) :byte[]
+ getAllowedType() :Class<Short>
HLAinteger32LECoder
- coder :HLAinteger32LE = null
- factory :EncoderFactory = null
+ HLAinteger32LECoder()
+ decode(byte[]) :Integer
+ encode(Integer) :byte[]
+ getAllowedType() :Class<Integer>
HLAinteger64LECoder
- coder :HLAinteger64LE = null
- factory :EncoderFactory = null
+ HLAinteger64LECoder()
+ decode(byte[]) :Long
+ encode(Long) :byte[]
+ getAllowedType() :Class<Long>
HLAbyteCoder
- coder :HLAbyte = null
- factory :EncoderFactory = null
+ HLAbyteCoder()
+ decode(byte[]) :Byte
+ encode(Byte) :byte[]
+ getAllowedType() :Class<Byte>
HLAbooleanCoder
- coder :HLAboolean = null
- factory :EncoderFactory = null
+ HLAbooleanCoder()
+ decode(byte[]) :Boolean
+ encode(Boolean) :byte[]
+ getAllowedType() :Class<Boolean>
«interface»
Coder
+ decode(byte[]) :T
+ encode(T) :byte[]
+ getAllowedType() :Class<T>
HLAasciiStringCoder
- coder :HLAASCIIstring = null
- factory :EncoderFactory = null
+ HLAasciiStringCoder()
+ decode(byte[]) :String
+ encode(String) :byte[]
+ getAllowedType() :Class<String>
HLAasciiCharCoder
- coder :HLAASCIIchar = null
- factory :EncoderFactory = null
+ HLAasciiCharCoder()
+ decode(byte[]) :Byte
+ encode(Byte) :byte[]
+ getAllowedType() :Class<Byte>
HLAunicodeCharCoder
- coder :HLAunicodeChar = null
- factory :EncoderFactory = null
+ HLAunicodeCharCoder()
+ decode(byte[]) :Short
+ encode(Short) :byte[]
+ getAllowedType() :Class<Short>
HLAunicodeStringCoder
- coder :HLAunicodeString = null
- factory :EncoderFactory = null
+ HLAunicodeStringCoder()
+ decode(byte[]) :String
+ encode(String) :byte[]
+ getAllowedType() :Class<String>
HLAfloat64LECoder
- coder :HLAfloat64LE = null
- factory :EncoderFactory = null
+ HLAfloat64LECoder()
+ decode(byte[]) :Double
+ encode(Double) :byte[]
+ getAllowedType() :Class<Double>
HLAfloat32LECoder
- coder :HLAfloat32LE = null
- factory :EncoderFactory = null
+ HLAfloat32LECoder()
+ decode(byte[]) :Float
+ encode(Float) :byte[]
+ getAllowedType() :Class<Float>
HLAfloat64BECoder
- coder :HLAfloat64BE = null
- factory :EncoderFactory = null
+ HLAfloat64BECoder()
+ decode(byte[]) :Double
+ encode(Double) :byte[]
+ getAllowedType() :Class<Double>
HLAfloat32BECoder
- coder :HLAfloat32BE = null
- factory :EncoderFactory = null
+ HLAfloat32BECoder()
+ decode(byte[]) :Float
+ encode(Float) :byte[]
+ getAllowedType() :Class<Float>

Figure 9: The UML class diagram of the skf.coder package.

4.1.6 The skf.utility package

The architecture of the skf.utility package is shown in Figure 10 by using a UML Class Diagram.

This package contains several miscellaneous utility classes. In particular, the *TimeUtility* class defines the mechanisms for the time standard conversions; and, the *SystemUtility* class provides a method to check the status of the MSWindows Firewall.

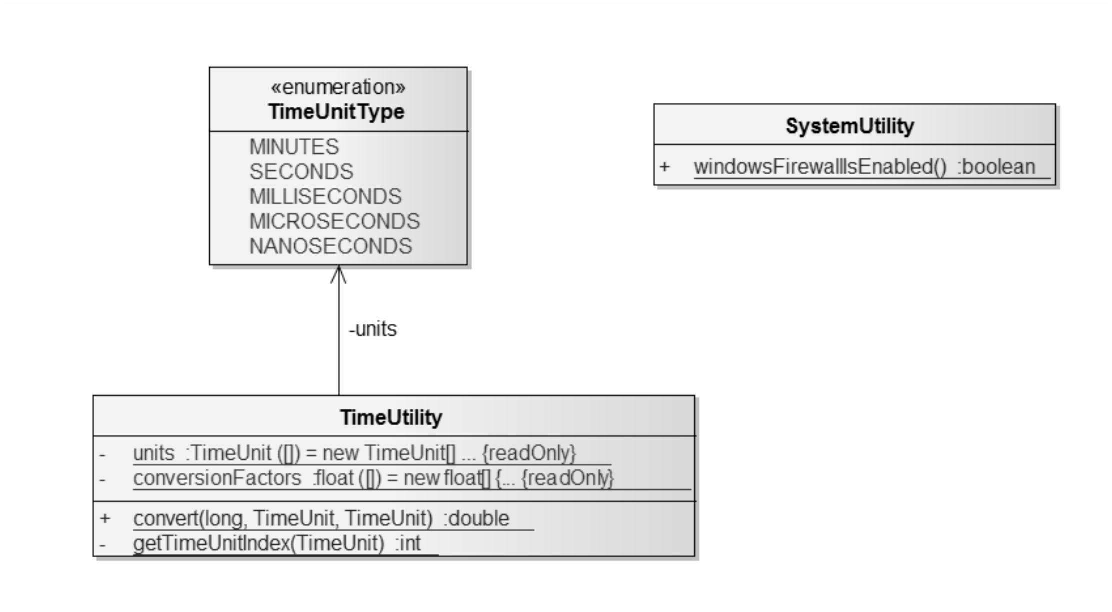
«enumeration»
TimeUnitType
MINUTES
SECONDS
MILLISECONDS
MICROSECONDS
NANOSECONDS
SystemUtility
+ windowsFirewallIsEnabled() :boolean
-units
TimeUtility
- units :TimeUnit ([]) = new TimeUnit[] ... {readOnly}
- conversionFactors :float ([]) = new float[] {... {readOnly}
+ convert(long, TimeUnit, TimeUnit) :double
- getTimeUnitIndex(TimeUnit) :int

Figure 10: The UML class diagram of the skf.utility package.

4.1.7 The skf.model package

The architecture of the skf.model package is shown in Figure 12 by using a UML Class Diagram.

This package contains classes to manage an *ObjectModel (ObjectClass* and *InteractionClass)*. In particular, the SKF framework uses these classes to facilitate the subscribe, publication and the data update of an *ObjectModel* provided by the user.

This package also includes a set of sub-packages each implementing a specific service.

4.1.7.1 The skf.model.parser package

The skf.model.parser defines the parser for the *ObjectModel (ObjectClass* and *InteractionClass)*. The parser is used by the SKF to determine the structure of the *ObjectModel*, in order to retrieve its structure and the values of its fields before publishing/updating on HLA/RTI platform. The architecture of the sub-package is shown in Figure 11 by using a UML Class Diagram.

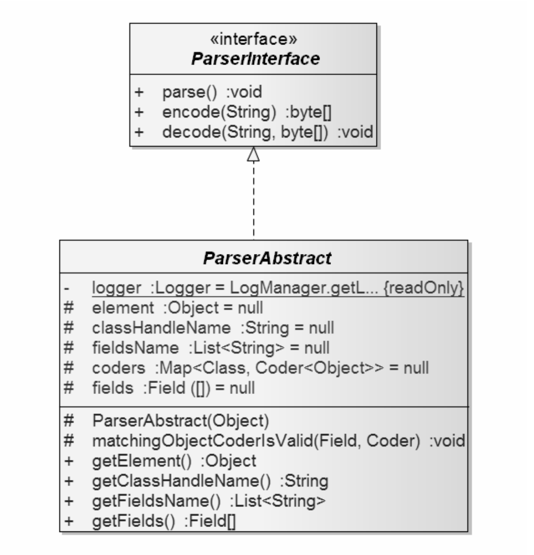
«interface»
ParserInterface
+ parse() :void
+ encode(String) :byte[]
+ decode(String, byte[]) :void
ParserAbstract
- logger :Logger = LogManager.getL... {readOnly}
# element :Object = null
# classHandleName :String = null
# fieldsName :List<String> = null
# coders :Map<Class, Coder<Object>> = null
# fields :Field ([]) = null
# ParserAbstract(Object)
# matchingObjectCoderIsValid(Field, Coder) :void
+ getElement() :Object
+ getClassHandleName() :String
+ getFieldsName() :List<String>
+ getFields() :Field[]

Figure 11: The UML class diagram of the skf.model.parser package.

4.1.7.2 The skf.model.object package

This package provides some classes to be used to define an *ObjectClass*; it uses the skf.model.object.annotation package (see Figure 13), which defines two *Annotation* interfaces, for creating an *ObjectClass*. In particular, these interfaces are used by the programmer to attach metadata to an ObjectClass class.

The architecture of the skf.model.object package is shown in Figure 12 by using a UML Class Diagram.

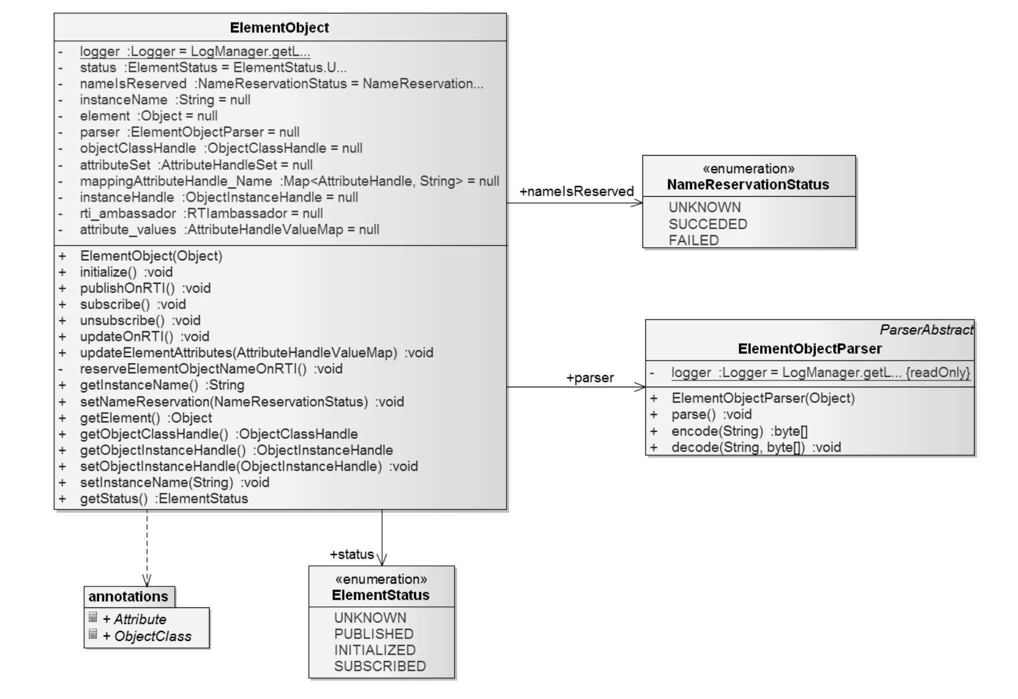
ElementObject
- logger :Logger = LogManager.getL...
- status :ElementStatus = ElementStatus.U...
- nameIsReserved :NameReservationStatus = NameReservation...
- instanceName :String = null
- element :Object = null
- parser :ElementObjectParser = null
- objectClassHandle :ObjectClassHandle = null
- attributeSet :AttributeHandleSet = null
- mappingAttributeHandle_Name :Map<AttributeHandle, String> = null
- instanceHandle :ObjectInstanceHandle = null
- rti_ambassador :RTIambassador = null
- attribute_values :AttributeHandleValueMap = null
+ ElementObject(Object)
+ initialize() :void
+ publishOnRTI() :void
+ subscribe() :void
+ unsubscribe() :void
+ updateOnRTI() :void
+ updateElementAttributes(AttributeHandleValueMap) :void
- reserveElementObjectNameOnRTI() :void
+ getInstanceName() :String
+ setNameReservation(NameReservationStatus) :void
+ getElement() :Object
+ getObjectClassHandle() :ObjectClassHandle
+ getObjectInstanceHandle() :ObjectInstanceHandle
+ setObjectInstanceHandle(ObjectInstanceHandle) :void
+ setInstanceName(String) :void
+ getStatus() :ElementStatus
+nameIsReserved
«enumeration»
NameReservationStatus
UNKNOWN
SUCCEDED
FAILED
ParserAbstract
ElementObjectParser
- logger :Logger = LogManager.getL... {readOnly}
+ ElementObjectParser(Object)
+ parse() :void
+ encode(String) :byte[]
+ decode(String, byte[]) :void
+parser
+status
annotations
+ Attribute
+ ObjectClass
«enumeration»
ElementStatus
UNKNOWN
PUBLISHED
INITIALIZED
SUBSCRIBED

Figure 12: The UML class diagram of the skf.model.object package.

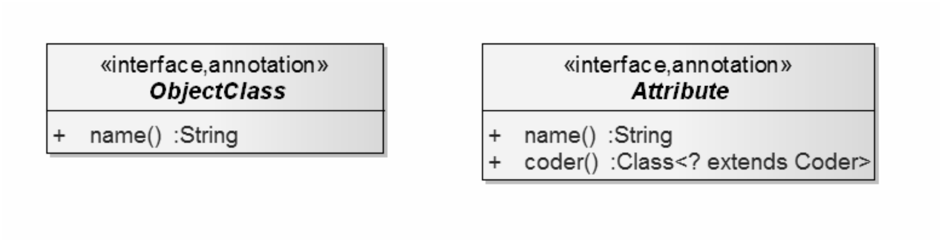

Figure 13: The UML class diagram of the skf.model.object.annotation package.

4.1.7.3 The skf.model.interaction package

It provides some classes used by the user to define an *InteractionClass*. It uses the skf.model.interaction.annotation package (see Figure 15), which defines two *Annotation* interfaces that are used by the programmer to attach metadata to an InterationClass class.

The architecture of the skf.model.interaction package is shown in Figure 14 by using a UML Class Diagram.

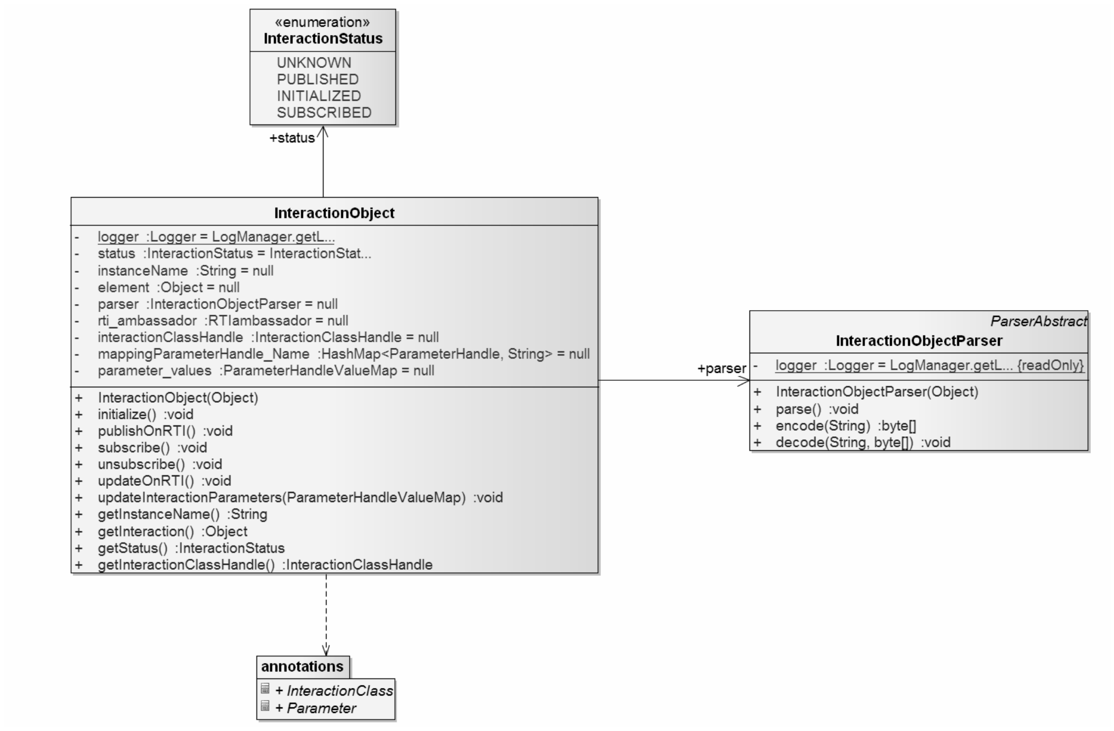
«enumeration»
InteractionStatus
UNKNOWN
PUBLISHED
INITIALIZED
SUBSCRIBED
+status
InteractionObject
- logger :Logger = LogManager.getL...
- status :InteractionStatus = InteractionStat...
- instanceName :String = null
- element :Object = null
- parser :InteractionObjectParser = null
- rti_ambassador :RTIambassador = null
- interactionClassHandle :InteractionClassHandle = null
- mappingParameterHandle_Name :HashMap<ParameterHandle, String> = null
- parameter_values :ParameterHandleValueMap = null
+ InteractionObject(Object)
+ initialize() :void
+ publishOnRTI() :void
+ subscribe() :void
+ unsubscribe() :void
+ updateOnRTI() :void
+ updateInteractionParameters(ParameterHandleValueMap) :void
+ getInstanceName() :String
+ getInteraction() :Object
+ getStatus() :InteractionStatus
+ getInteractionClassHandle() :InteractionClassHandle
+parser
ParserAbstract
InteractionObjectParser
- logger :Logger = LogManager.getL... {readOnly}
+ InteractionObjectParser(Object)
+ parse() :void
+ encode(String) :byte[]
+ decode(String, byte[]) :void
annotations
+ InteractionClass
+ Parameter

Figure 14: The UML class diagram of the skf.model.interaction package.

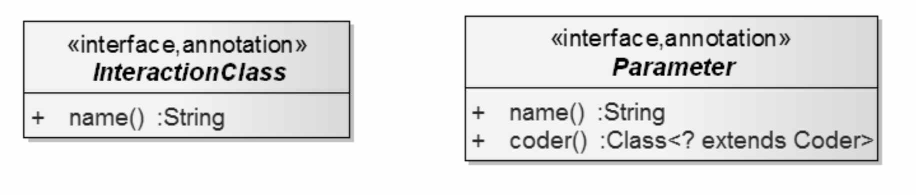

Figure 15: The UML class diagram of the skf.model.interaction.annotation package.

Appendix A: Starter Kit Framework (SKF) Release Roadmap

This section describes the medium-term roadmap for the Starter Kit Framework (SKF) project.

The release plan described below is still preliminary and might change.

Feature Release: Version 1.0 **Release Date: 18/07/2014**
• **CONNECTION MANAGEMENT services** o Added the mechanisms to manage the connection (set-up, hold-up and close-up) of a SEE Federate on the RTI.
• **FOM MANAGEMENT services** o Added the mechanisms to facilitate the management and the publication of the FOM modules.
• **INTERACTION MANAGEMENT services**
• **SIMULATION TIME MANAGEMENT services**
• **COORDINATES FRAME MANAGEMENT services**
• **LOGGING services**
• **TESTING services**
• **OWNERSHIP TRANSFER and DATA DISTRIBUTION MANAGEMENT services:**

Feature Release: Version 2.0	**Release Date:01/09/2014**
• **CONNECTION MANAGEMENT services**	
• **FOM MANAGEMENT services**	
• **INTERACTION MANAGEMENT services** o Added the mechanisms to facilitate the publish and the update of the module's attributes	
• **SIMULATION TIME MANAGEMENT services** o Added the mechanisms to manage the simulation time.	
• **COORDINATES FRAME MANAGEMENT services**	
• **LOGGING services**	
• **TESTING services**	
• **OWNERSHIP TRANSFER and DATA DISTRIBUTION MANAGEMENT services:**	

Feature Release: Version 3.0	**Release Date: 01/10/2014**
• **CONNECTION MANAGEMENT services**	
• **FOM MANAGEMENT services**	
• **INTERACTION MANAGEMENT services**	

• **SIMULATION TIME MANAGEMENT services**   ○ Added the mechanisms for time standard conversions.
• **COORDINATES FRAME MANAGEMENT services**   ○ Added the mechanisms to manage the transformations among SEE Coordinate Systems.   ○ Added the functionality for publication of SEE Reference Frames.
• **LOGGING services**
• **TESTING services**   ○ Check the IP Configuration.   ○ Check the MS Windows Firewall.
• **OWNERSHIP TRANSFER and DATA DISTRIBUTION MANAGEMENT services:**

Appendix B: Starter Kit Framework (SKF) features

The following table shows a summary of the SKF features.

Feature	Available
Mechanisms to manage the connection (set-up, hold-up and close-up) of a SEE Federate on the RTI.	☒
Mechanisms to facilitate the management and the publication of FOM modules.	☒
Mechanisms to facilitate the management of the configuration parameters.	☒
Mechanisms to facilitate publishing and updating of information on the RTI.	☒
Mechanisms to manage the simulation time.	☒
Mechanisms for time standard conversions.	☒
Mechanisms to manage the transformations among SEE Coordinate Systems.	☐
Functionalities for subscribing SEE Reference Frames.	☒
Check of the IP Configuration.	☒
Check of the MS Windows Firewall state.	☒
Logging.	☒
Ownership transfer and data distribution management.	☐

Appendix C: Using the SKF framework: the "TestFederate"

The SKF framework has been experimented recreating the "EnvironmentTest" Federate provided by the NASA team.

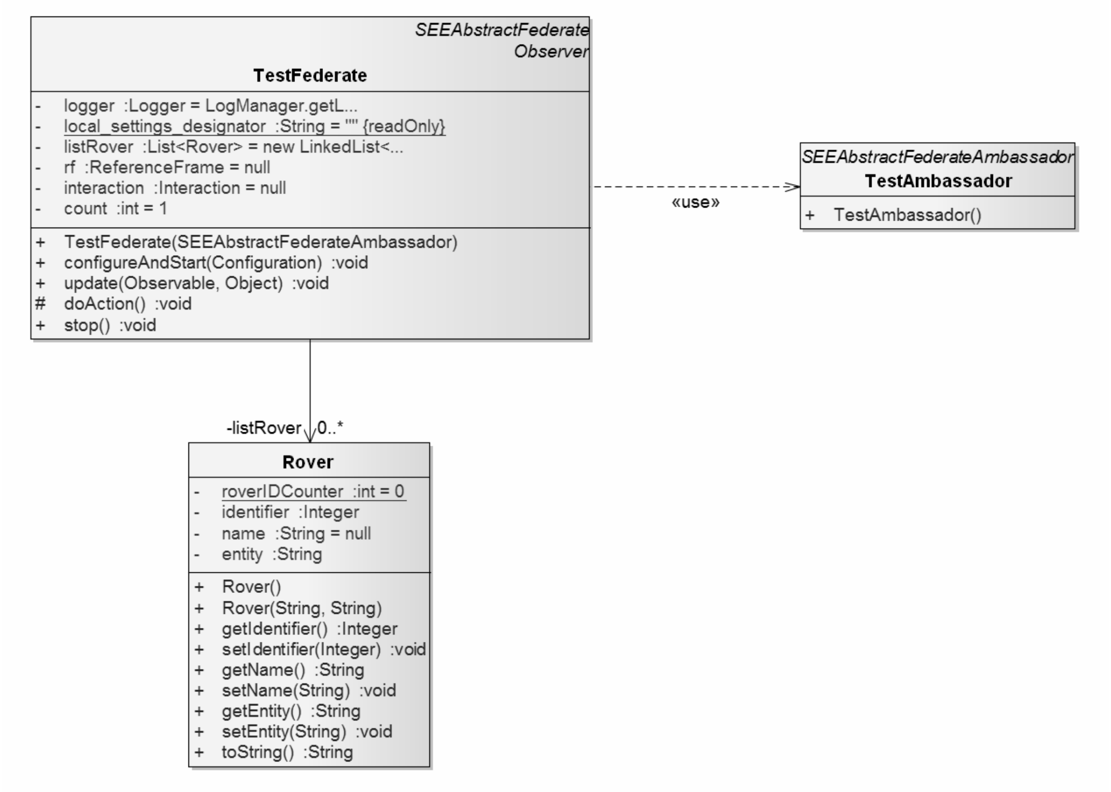
SEEAbstractFederate
Observer
TestFederate
- logger :Logger = LogManager.getL...
- local_settings_designator :String = "" {readOnly}
- listRover :List<Rover> = new LinkedList<...
- rf :ReferenceFrame = null
- interaction :Interaction = null
- count :int = 1
+ TestFederate(SEEAbstractFederateAmbassador)
+ configureAndStart(Configuration) :void
+ update(Observable, Object) :void
# doAction() :void
+ stop() :void
«use»
SEEAbstractFederateAmbassador
TestAmbassador
+ TestAmbassador()
-listRover
0..*
Rover
- roverIDCounter :int = 0
- identifier :Integer
- name :String = null
- entity :String
+ Rover()
+ Rover(String, String)
+ getIdentifier() :Integer
+ setIdentifier(Integer) :void
+ getName() :String
+ setName(String) :void
+ getEntity() :String
+ setEntity(String) :void
+ toString() :String

Figure 16: The UML class diagram of the "TestFederate".

The architecture of the *TestFederate* is shown in Figure 16 by using a UML Class Diagram.

The Java code of the classes related to the *TestFederate* is reported in the following.

The TestFederate class

```
package federate;

import hla.rti1516e.exceptions.*;
import java.net.MalformedURLException;
import java.util.LinkedList;
import java.util.List;
import java.util.Observable;
import java.util.Observer;
import java.util.Scanner;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

import model.Rover;
import model.Interaction;
import siso.smackdown.FrameType;
import siso.smackdown.ReferenceFrame;
import skf.config.Configuration;
import skf.core.SEEAbstractFederate;
import skf.core.SEEAbstractFederateAmbassador;
import skf.exception.UnsubscribeException;
import skf.exception.UpdateException;
import skf.utility.JulianDateType;
import skf.utility.TimeUnit;
import skf.utility.TimeUtility;


public class TestFederate extends SEEAbstractFederate implements Observer {

        private Logger logger = LogManager.getLogger(TestFederate.class);

        /*
         * For MAK local_settings_designator = "";
         * For PITCH local_settings_designator = "crcHost=" + <crc_host> + "\ncrcPort=" + <crc_port>;
         */
        private static final String local_settings_designator = "";

        private List<Rover> listRover = new LinkedList<Rover>();
        private ReferenceFrame rf = null;
        private Interaction interaction = null;

        public TestFederate(SEEAbstractFederateAmbassador seefedamb) {
                super(seefedamb);
        }
```

```
	public void configureAndStart(Configuration config) throws Exception {

		// 1. configure the SKF framework
		super.configure(config);
		// 2. Connect on RTI
		super.connectOnRTI(local_settings_designator);
		// 3. The Federate joins into the Federation execution
		super.joinIntoFederationExecution();


		try {
			// 4. Publish/Subscribe
			super.subscribeSubject(this);
			// 5. Publish/Subscribe objects and/or interaction
			super.subscribeReferenceFrame(FrameType.MarsCentricInertial);
			super.subscribeReferenceFrame(FrameType.EarthCentricFixed);
			listRover.add(new Rover("lunarRover1","lunarRover1"));
			listRover.add(new Rover("lunarRover2","lunarRover2"));

			for(Rover r : listRover)
				super.publishElement(r, r. getName());

			interaction = new Interaction("interaction", "payload");
			super.publishInteraction(interaction);

		} catch (Exception e) {
			e.printStackTrace();
		}

		// 6. Execution-loop
		super.startExecution();

		try {
			logger.info("Press any key to disconnect the federate from the federation
execution");
			new Scanner(System.in).next();
			stop();
		} catch (Exception e) {
			e.printStackTrace();
		}
	}

	@Override
	public void update(Observable o, Object arg) {
		this.rf = (ReferenceFrame) arg;
		logger.info(rf);

	}

	private int count = 1;
```

```
        @Override
        protected void doAction() {
                logger.info("federation execution time cycle: 
"+TimeUtility.convert(super.getTime().getFederationExecutionTimeCycle(), TimeUnit.MICROSECONDS, 
TimeUnit.SECONDS));
                logger.info("federate time cycle: 
"+TimeUtility.convert(super.getTime().getFederateExecutionTimeCycle(), TimeUnit.MICROSECONDS, 
TimeUnit.SECONDS));

                logger.info("federate time date: "+super.getTime().getFederationExecutionTime());

                logger.info("federation execution jd: 
"+super.getTime().getFederationExecutionTimeInJulianDate(JulianDateType.DATE));
                logger.info("federation execution mjd: 
"+super.getTime().getFederationExecutionTimeInJulianDate(JulianDateType.MODIFIED));
                logger.info("federation execution rjd: 
"+super.getTime().getFederationExecutionTimeInJulianDate(JulianDateType.REDUCED));
                logger.info("federation execution tjd: 
"+super.getTime().getFederationExecutionTimeInJulianDate(JulianDateType.TRUNCATED));

                try {

                        Rover tmp = null;

                        for(int i=0; i<listRover.size(); i++){
                                tmp = listRover.get(i);
                                // update the Identifier of the Rover
                                tmp.setIdentifier(count);
                                // update on RTI the rover
                                super.updateElement(tmp);
                                count++;
                        }

                        //update the 'payload' of the RoverInteraction
                        interaction.setPayload("payload"+Math.random());
                        super.updateInteraction(interaction);


                } catch (Exception e) {
                        e.printStackTrace();
                }

        }

        public void stop() throws Exception {

                // Unsubcribe from the Subject
                super.unsubscribeSubject(this);

                // Disconnect from the HLA/RTI platform
                super.diconnectFromRTI();
        }

}
```

The TestAmbassador class

```
import skf.core.SEEAbstractFederateAmbassador;

public class TestAmbassador extends SEEAbstractFederateAmbassador {

        public TestAmbassador() {
                super();
        }
}
```

The Main class

```
import java.io.File;
import java.io.FileNotFoundException;

import federate.TestAmbassador;
import federate.TestFederate;
import skf.config.Configuration;
import skf.config.ConfigurationFactory;


public class MainTest {

        private static final File configurationFile = new File("testResources/configuration/conf.json");

        public static void main(String[] args) throws FileNotFoundException {

                TestAmbassador testfedamb = new TestAmbassador();
                TestFederate testfed = new TestFederate(testfedamb);
                ConfigurationFactory factory = null;
                Configuration config = null;
                try {
                        factory = new ConfigurationFactory();
                        config = factory.importConfiguration(configurationFile);
                        testfed.configureAndStart(config);

                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

The Rover class (ObjectClass)

```
(note [1])
package model;

import skf.coder.HLAinteger32BECoder;
import skf.coder.HLAunicodeStringCoder;
import skf.model .annotations.Attribute;
import skf.model .annotations.ObjectClass;

@ObjectClass(name = "PhysicalEntity.Rover") (note [2])
public class Rover {

        private static int roverIDCounter = 0;

        @Attribute(name = "id", coder = HLAinteger32BECoder.class)
        private Integer identifier;

        @Attribute(name = "name", coder = HLAunicodeStringCoder.class)
        private String name = null;

        public Rover() {}

        public Rover(String name) {
                this.identifier = roverIDCounter;
                roverIDCounter++;
                this.name = name;
        }
        public Integer getIdentifier() {
                return identifier;
        }
        public void setIdentifier(Integer identifier) {
                this.identifier = identifier;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
        @Override
        public String toString() {
                return "Rover [identifier=" + identifier + ", name=" + name + "]";
        }
}
```

The Interaction class (InteractionClass)

```
(note [1])
package model;

import skf.coder.HLAunicodeStringCoder;
import skf.model.interaction.annotations.InteractionClass;
import skf.model.interaction.annotations.Parameter;


@InteractionClass(name = "RoverInteraction") (note [3])
public class Interaction {

	@Parameter(name = "name", coder = HLAunicodeStringCoder.class)
	private String name = null;

	@Parameter(name = "payload", coder = HLAunicodeStringCoder.class)
	private String payload = null;

	public Interaction() {}

	public Interaction(String name, String payload) {
		this.name = name;
		this.payload = payload;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getPayload() {
		return payload;
	}

	public void setPayload(String payload) {
		this.payload = payload;
	}

	@Override
```

```
        public String toString() {
                return "Interaction [name=" + name + ", payload=" + payload + "]";
        }

}
```

NOTE:

[1] In order to be processed by the SEE Starter Kit Framework, the ObjectClass/InteractionClass class must be a JavaBean. It is a specially constructed Java class coded according to the JavaBeans API specifications (for more details see the website: http://download.oracle.com/otndocs/jcp/7224-javabeans-1.01-fr-spec-oth-JSpec/). The main characteristics that distinguish a JavaBean from other Java classes are the following:

- A JavaBean provides a default, no-argument constructor;
- A JavaBean should be serializable and implement the Serializable interface. (this characteristic is no necessary for the Kit);
- A JavaBean may have a number of properties which can be read or written;
- A JavaBean may have a number of "getter" and "setter" methods for the properties.

[2] The value of the attribute ‘name’ must match with the path defined in its FOM file, starting from the tag <name>HLAobjectRoot</name> and with the root not included (in this case, PhysicalEntity.Rover).

[3] The value of the attribute ‘name’ must match with the path defined in its FOM file, starting from the tag <name>HLAinteractionRoot</name> and with the root not included (in this case, RoverInteraction).

The Rover FOM

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<objectModel
	xsi:schemaLocation="http://standards.ieee.org/IEEE1516-2010
http://standards.ieee.org/downloads/1516/1516.2-2010/IEEE1516-DIF-2010.xsd"
	xmlns="http://standards.ieee.org/IEEE1516-2010"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
	<modelIdentification>
		<name>MyModule</name>
		<type>FOM</type>
		<version>1.0</version>
		<securityClassification>unclassified</securityClassification>
		<purpose></purpose>
		<applicationDomain></applicationDomain>
		<description>Description of MyModule</description>
		<useLimitation></useLimitation>
		<other></other>
	</modelIdentification>
	<objects>
		<objectClass>
			<name>HLAobjectRoot</name>
			<objectClass>
				<name>PhysicalEntity</name>
				<objectClass>
					<name>Rover</name>
					<sharing>PublishSubscribe</sharing>
					<semantics></semantics>
					<attribute>
						<name>id</name>
						<dataType>HLAinteger16BE</dataType>
						<updateType>Static</updateType>
						<updateCondition>NA</updateCondition>
						<ownership>NoTransfer</ownership>
						<sharing>PublishSubscribe</sharing>
						<transportation>HLAreliable</transportation>
						<order>Receive</order>
						<semantics>The identifier of the ROVER
							module.</semantics>
					</attribute>
					<attribute>
						<name>name</name>
						<dataType>HLAunicodeString</dataType>
						<updateType>Static</updateType>
						<updateCondition>NA</updateCondition>
						<ownership>NoTransfer</ownership>
						<sharing>PublishSubscribe</sharing>
						<transportation>HLAreliable</transportation>
						<order>Receive</order>
						<semantics>The name of the ROVER module.</semantics>
```

```
                        </attribute>
                    </objectClass>
                </objectClass>
            </objectClass>
        </objects>
        <interactions>
            <interactionClass>
                <name>HLAinteractionRoot</name>
                <interactionClass>
                    <name>RoverInteraction</name>
                    <sharing>PublishSubscribe</sharing>
                    <transportation>HLAreliable</transportation>
                    <order>Receive</order>
                    <semantics></semantics>
                    <parameter>
                        <name>name</name>
                        <dataType>HLAunicodeString</dataType>
                        <semantics></semantics>
                    </parameter>
                    <parameter>
                        <name>payload</name>
                        <dataType>HLAunicodeString</dataType>
                        <semantics></semantics>
                    </parameter>
                </interactionClass>
            </interactionClass>
        </interactions>
        <dataTypes>
            <simpleDataTypes />
            <enumeratedDataTypes />
            <arrayDataTypes />
            <fixedRecordDataTypes />
            <variantRecordDataTypes />
        </dataTypes>
        <notes />
</objectModel>
```

The Configuration file

```
{
 "asynchronousDelivery" : true,
 "crcHost" : "localhost",
 "crcPort" : 8989,
 "federateName" : "SEETestFederate",
 "federateType" : "Test",
 "federationName" : "SEE 2015",
 "fomDirectory" : "C: \\foms",
 "realtime" : false,
 "simulationEphoc" : "2015-04-12T20:00:00.000Z", (note [4])
 "timeConstrained" : true,
 "timeRegulating" : false
}
```

NOTE:

[4] The Simulation Ephoc is expressed according to the ISO 8601 standard, for more detail see the website: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=40874.

For the SEE 2015 event, the Simulation Ephoc has been set to 04/12/2015 20:00:00 GMT.