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 B), 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).

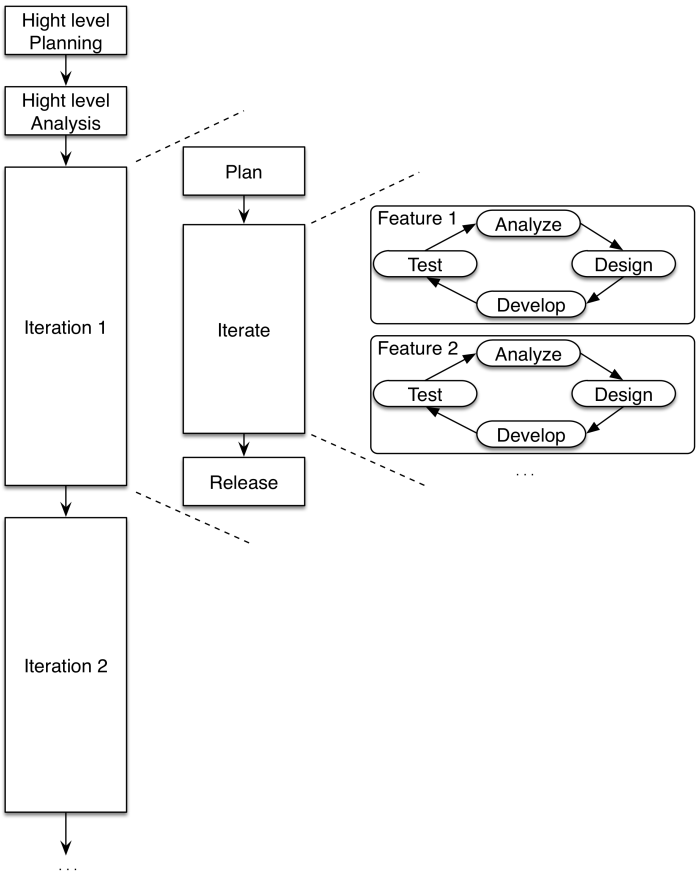

Figure 1: The exploited Agile software development process.

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;
- Fully-Support to the SISO SPACE Reference FOM ongoing standard;
- ...;

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.

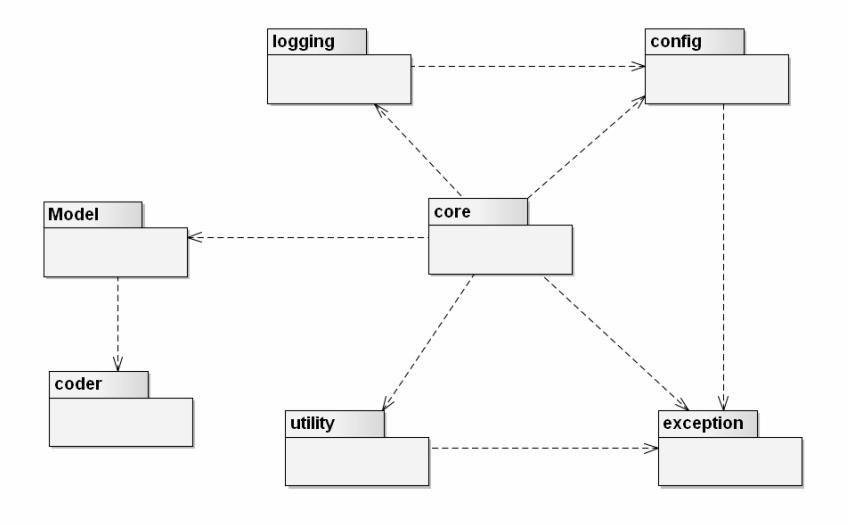

Figure 2: SEE HLA Starter Kit, UML Package Diagram.

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.

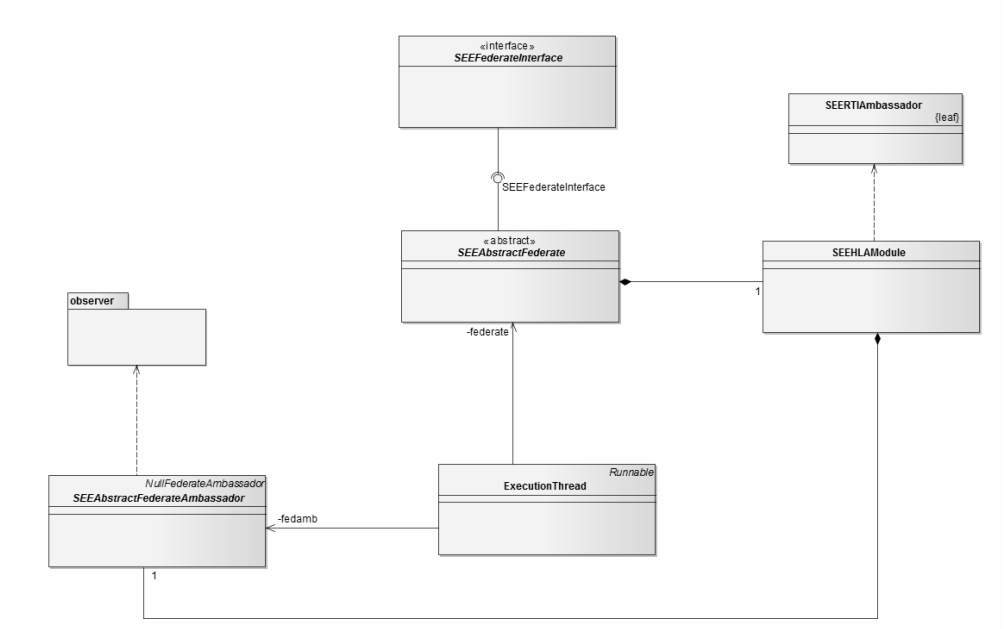

Figure 3 : The UML class diagram of the skf.core package.

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".

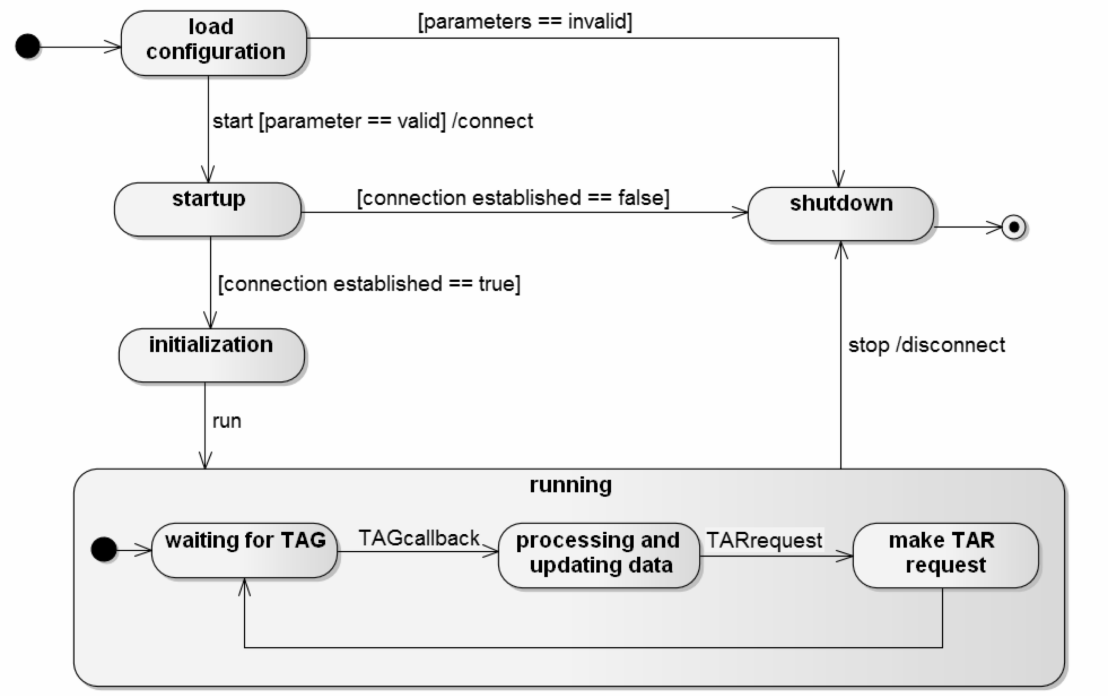

Figure 4: a UML statechart of the life cycle.

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.

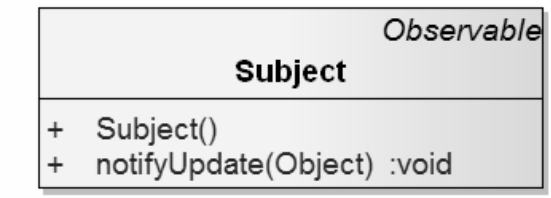

Figure 5: The UML class diagram of the skf.config.observer package.

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.

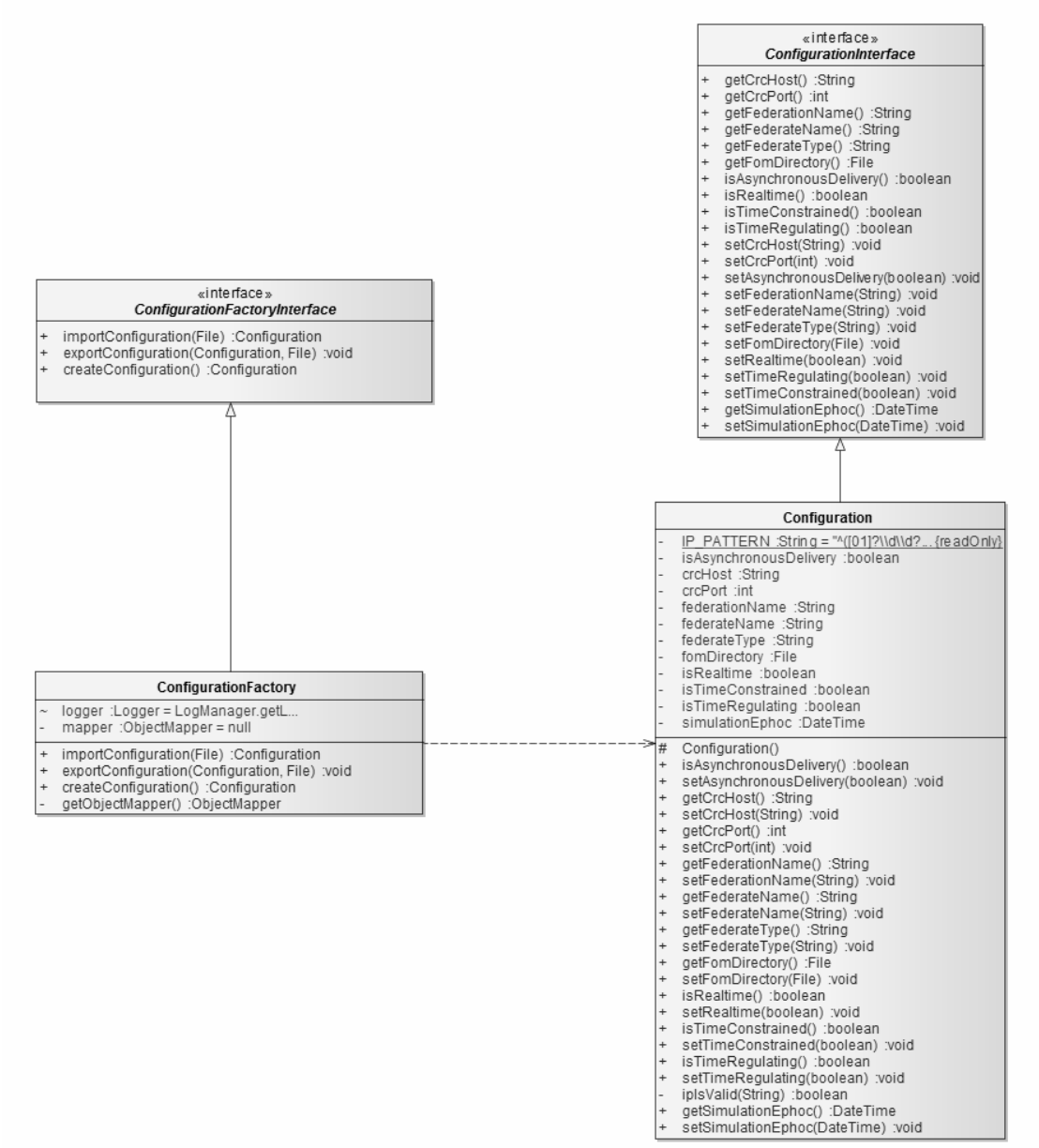

Figure 6: The UML class diagram of the skf.config package.

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.

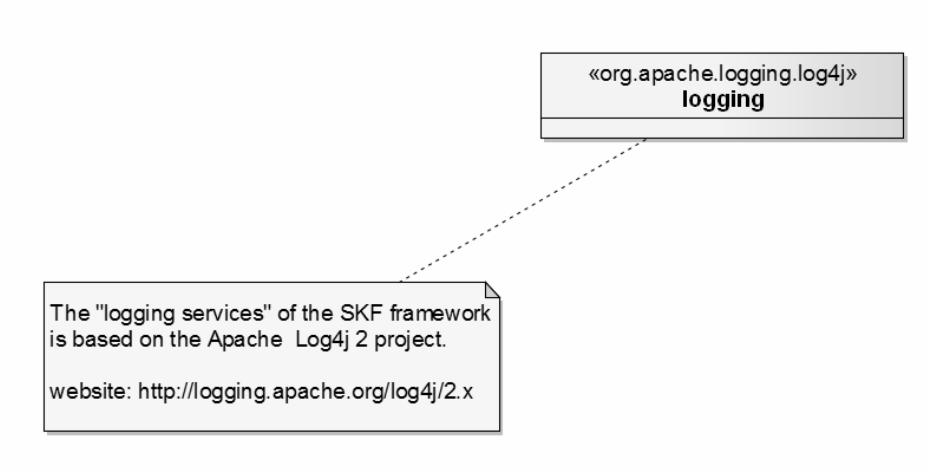

Figure 7: The UML class diagram of the skf.logging package.

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.

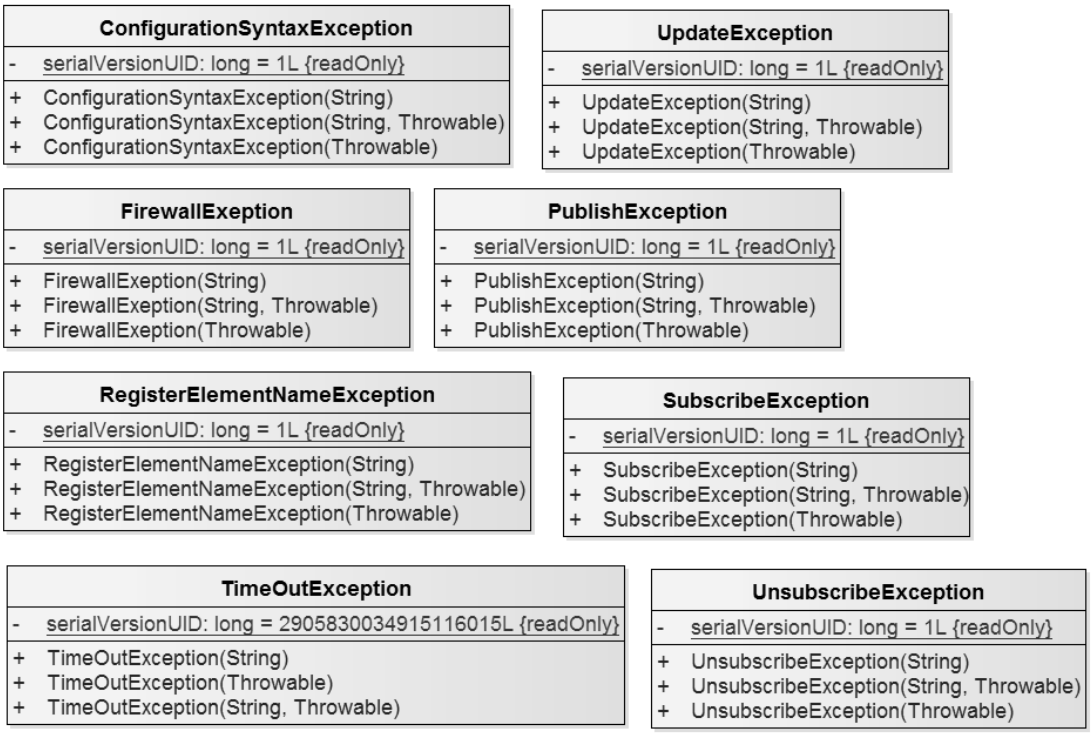

Figure 8: The UML class diagram of the skf.exception package.

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.

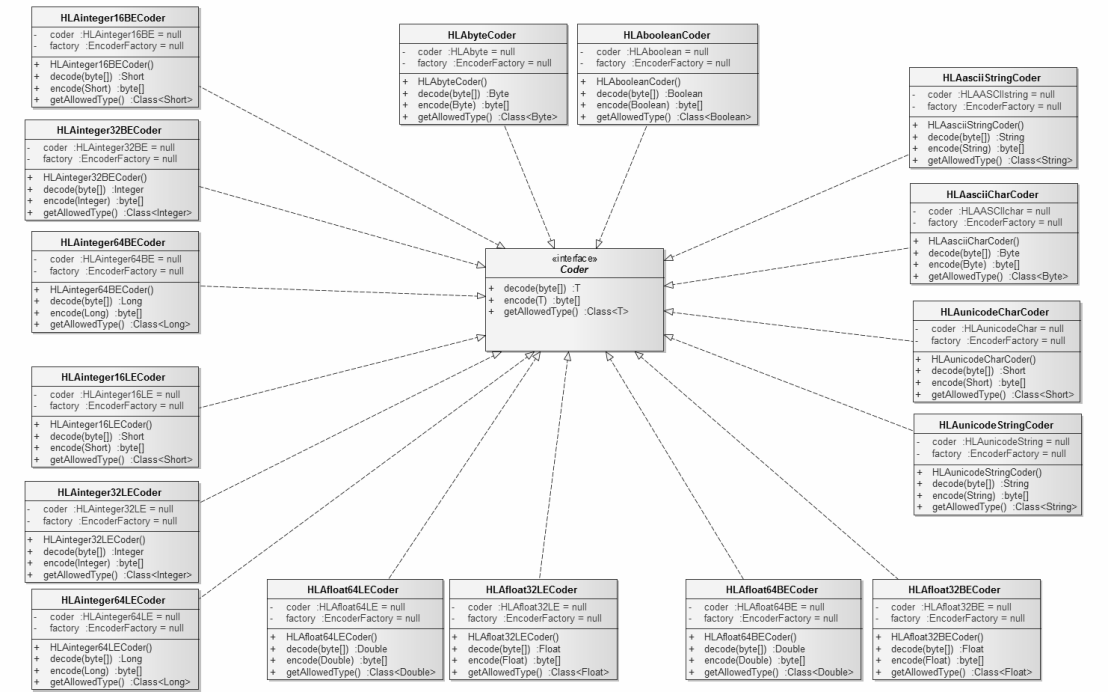

Figure 9: The UML class diagram of the skf.coder package.

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.

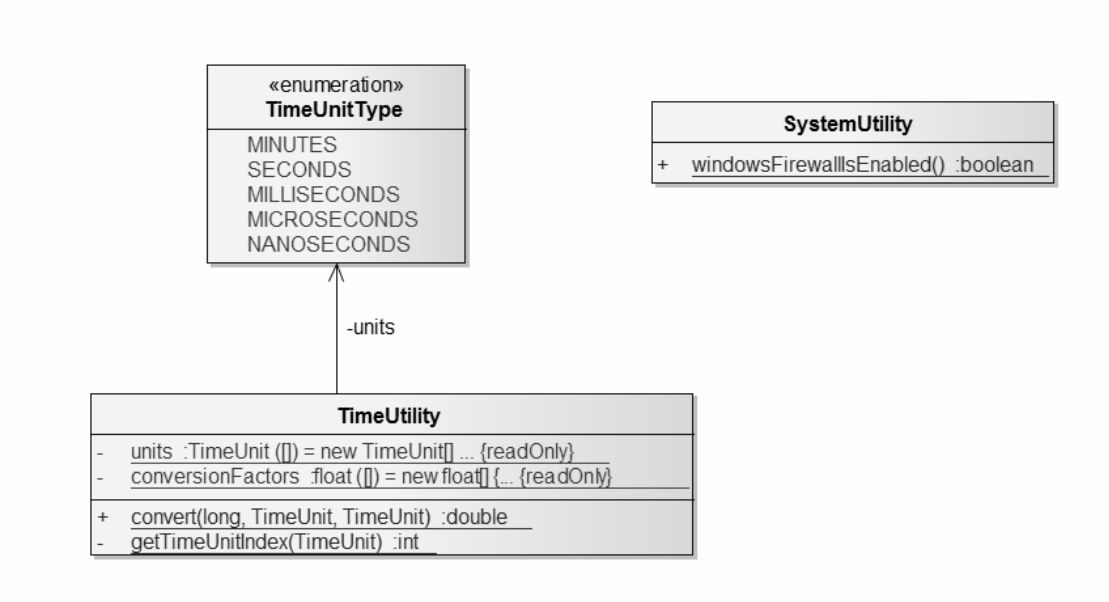

Figure 10: The UML class diagram of the skf.utility package.

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.

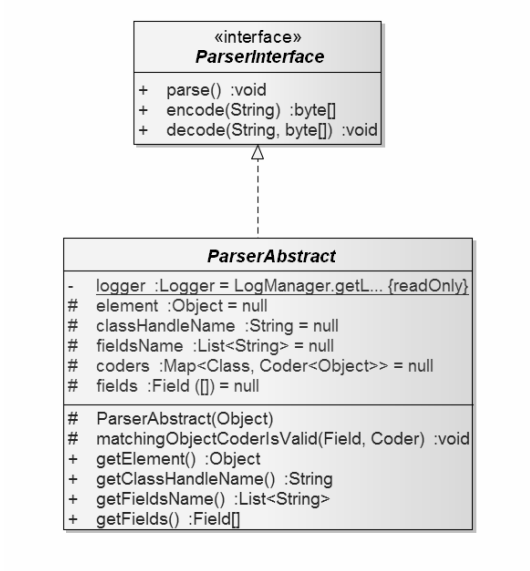

Figure 11: The UML class diagram of the skf.model.parser package.

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.

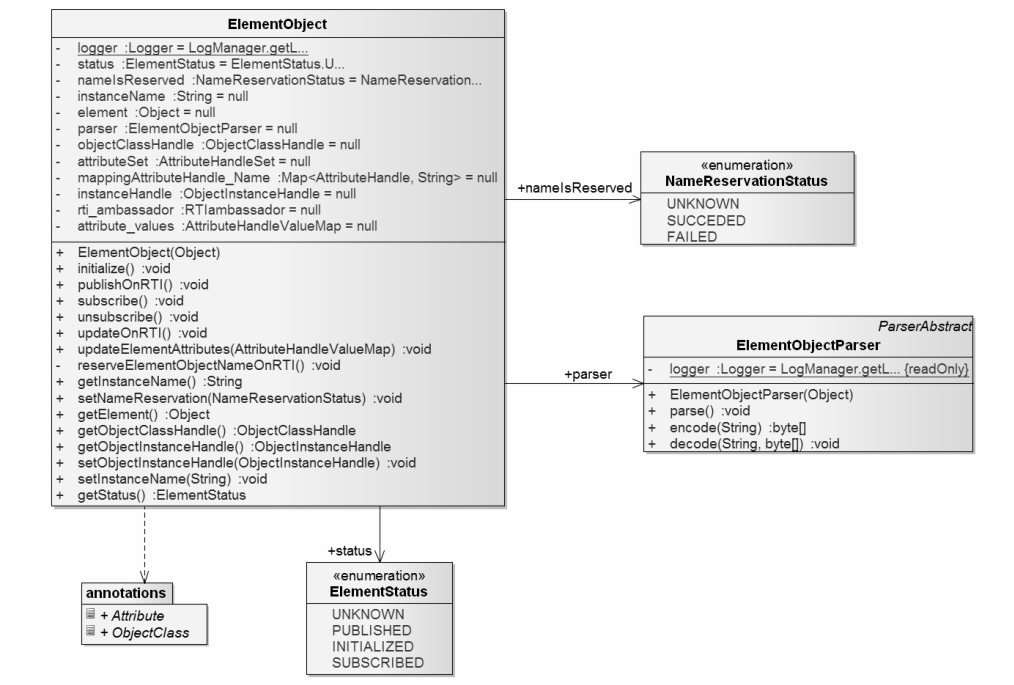

Figure 12: The UML class diagram of the skf.model.object package.

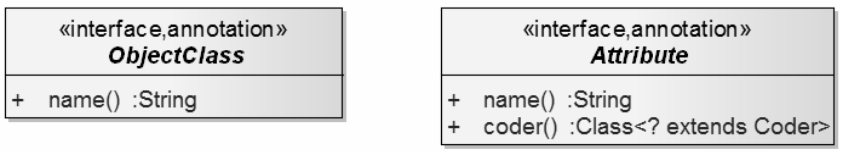

Figure 13: The UML class diagram of the skf.model.object.annotation package.

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.

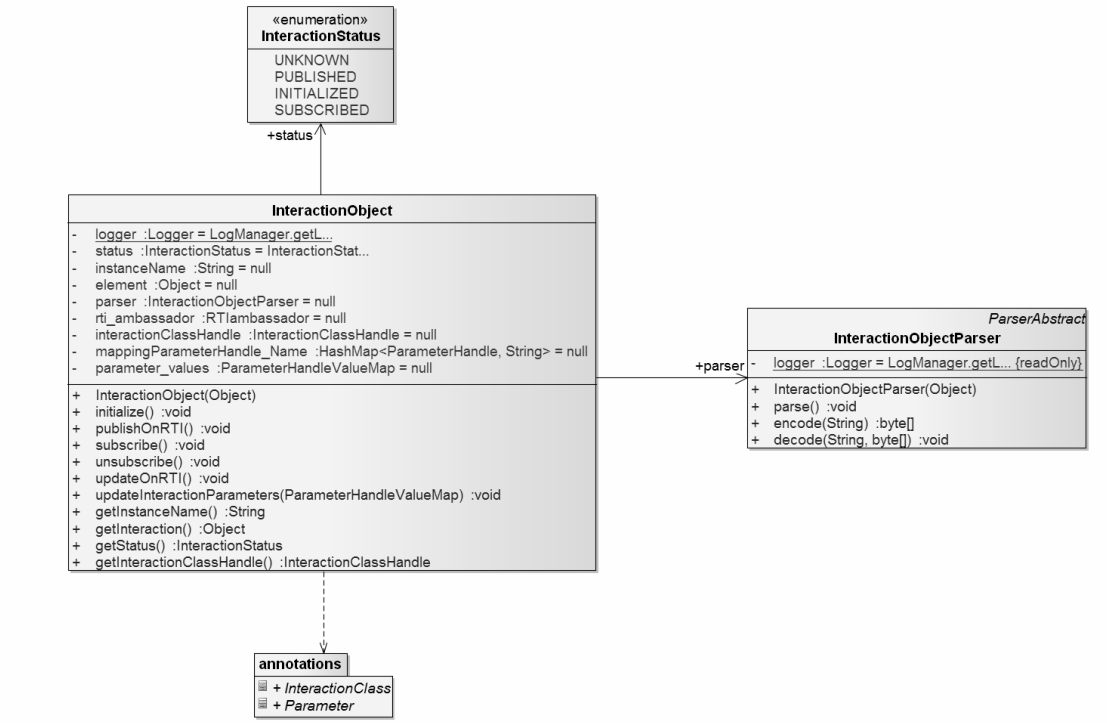

Figure 14: The UML class diagram of the skf.model.interaction package.

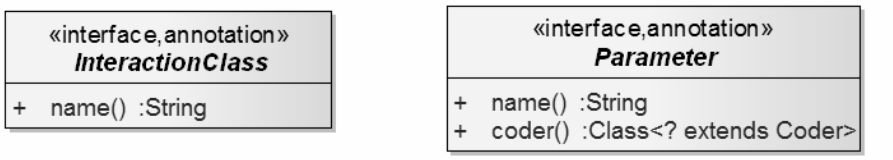

Figure 15: The UML class diagram of the skf.model.interaction.annotation package.

Appendix A: Starter Kit Framework (SKF) 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 manage the SISO SPACE Reference FOM ongoing standard.	☒
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 B: Using the SKF framework: the "TestFederate"

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

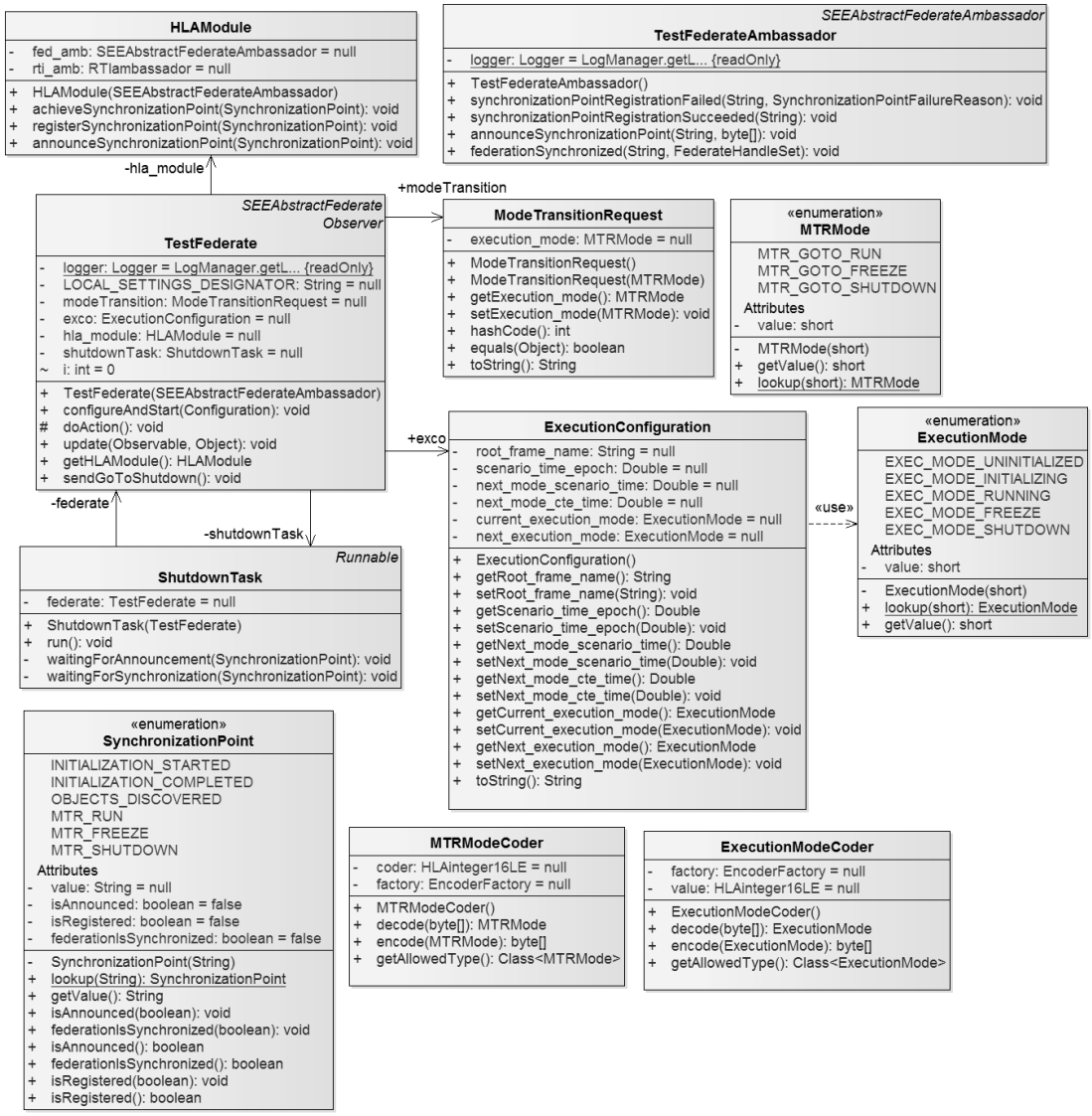

Figure 16: The UML class diagram of the "TestFederate".

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.*;
import model.interactionClass.*;
import model.objectClass.*;
import org.apache.logging.log4j.*;
import siso.smackdown.frame.*;
import skf.config.Configuration;
import skf.core.*;
import skf.exception.*;

public class TestFederate extends SEEAbstractFederate implements Observer
{

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

	private String LOCAL_SETTINGS_DESIGNATOR = null;
	private ModeTransitionRequest modeTransition = null;
	private ExecutionConfiguration exco = null;
	private HLAModule hla_module = null;
	private ShutdownTask shutdownTask = null;

	public TestFederate(SEEAbstractFederateAmbassador seefedamb) throws
RTIinternalError {
		super(seefedamb);
		this.modeTransition = new ModeTransitionRequest();
		this.exco = new ExecutionConfiguration();
		this.hla_module  = new HLAModule(seefedamb);
		this.shutdownTask = new ShutdownTask(this);
	}

	public void configureAndStart(Configuration config) throws Exception
{
// -------------------- Federation Join Process -------------------- //
		/*
		*  1. Start
		*  Configure the SKF core components
		*/
		super.configure(config);
		super.subscribeSubject(this);

		/*
		* 2. Connect to RTI
		*
		* For VT MAK LOCAL_SETTINGS_DESIGNATOR = "";
		* For Pitch RTI LOCAL_SETTINGS_DESIGNATOR = "crcHost=" +
<crcHost> + "\ncrcPort=" + <crcPort>;
```

```
            */
            LOCAL_SETTINGS_DESIGNATOR =
"crcHost="+config.getCrcHost()+"\ncrcPort="+config.getCrcPort();
            super.connectOnRTI(LOCAL_SETTINGS_DESIGNATOR);

            /**
            * 3. Join the Federation Execution
            */
            super.joinIntoFederationExecution();

            /**
            * 4. Enable Asynchronous Delivery
            * Already done through the use of the 'configuration.json'
file.
            * Please make sure that in the 'configuration.json' file the
'asynchronousDelivery' attribute is set to 'true'
            */

// -------------------- Federate Initialization Process ----------------
-- //

            /**
            * 1. Setup RTI Handles
            * Not needed because it is handled by the skf core components
            */

            /**
            * 2. Subscribe Execution Control Object (ExCO) Class
Attributes
            */
            super.subscribeElement(ExecutionConfiguration.class);

            /**
            * 3. Wait for ExCO Discovery
            */
            super.waitForElementDiscovery(ExecutionConfiguration.class);

            /*
            * 4. Request ExCO Update
            */
      super.requestAttributeValueUpdate(ExecutionConfiguration.class);

            /**
            * 5. Wait for ExCO Update and Check for Mode Transition to
Shutdown.
            * The callback is managed by the 'update' method
            */
      super.waitForAttributeValueUpdate(ExecutionConfiguration.class);

            /**
            * 6. Publish Mode Transition Request (MTR) Interaction
            */
            super.publishInteraction(modeTransition);
```

```
            /**
            * 7. Publish and Subscribe Federate Object Class Attributes
and Interaction Classes
            * 8. Reserve All Federate Object Instance Names
            * 9. Wait for All Federate Object Instance Name Reservation
Success/Failure Callbacks
            * 10. Register Federate Object Instances
            * 11. Wait for All Required Objects to be Discovered
            *
            * Use here:
            * -  super.subscribeElement(objectClass);
            * -  super.subscribeInteraction(interactionClass);
            */
            super.subscribeReferenceFrame(FrameType.MoonCentricFixed);

            /**
            * 12. Setup HLA Time Management: Enable Time Constrained
Disable Time Regulating
            * Already done by using the 'configuration.json' file.
            * Please make sure that in the 'configuration.json' file the:
            * - 'timeConstrained' attribute is set to 'true'
            * - 'timeRegulating' attribute is set to 'false'
            */

            /**
            * 13. Query GALT and Time Advance to GALT
            * Already done by the skf core components
            */

            /**
            * 14. Goto Execution
            */
            super.startExecution();


      }

      int i = 0;
      @Override
      protected void doAction() {
            /*
            *  proactive behavior
            */
            logger.log(Level.INFO, "step: " + (i++));

      }

      @Override
      public void update(Observable o, Object arg) {

            /*
            *  reactive behavior
```

```
            */
      if(arg instanceof ExecutionConfiguration){
            logger.log(Level.INFO, "**** Update ExecutionConfiguration
****");
            exco = ((ExecutionConfiguration)arg);
            if(exco.getNext_execution_mode() ==
ExecutionMode.EXEC_MODE_SHUTDOWN)
                  new Thread(shutdownTask).start();
            }

      if(arg instanceof ReferenceFrame){
            logger.log(Level.INFO, "**** Update ReferenceFrame ****");
            logger.log(Level.INFO, (ReferenceFrame)arg);
         }
      }

      public HLAModule getHLAModule() {
            return this.hla_module;
      }

      public void sendGoToShutdown() {

            modeTransition.setExecution_mode(MTRMode.MTR_GOTO_SHUTDOWN);
            try {
                  super.updateInteraction(modeTransition);
            } catch (Exception e) {e.printStackTrace();}
      }
}
```

The TestFederateAmbassador class

```
package federate;

import hla.rti1516e.*;
import org.apache.*;
import skf.core.SEEAbstractFederateAmbassador;
import synchronizationPoint.SynchronizationPoint;

public class TestFederateAmbassador extends SEEAbstractFederateAmbassador {

      private final static Logger logger = LogManager.getLogger(TestFederateAmbassador.class);


      public TestFederateAmbassador() {
            super();
      }

      // ---------------------- SynchronizationPoint -----------------------
      @Override
      public void synchronizationPointRegistrationFailed(String label, SynchronizationPointFailureReason
```

```
reason ) {
		SynchronizationPoint sp = SynchronizationPoint.lookup(label);
		if(sp != null)
			System.out.println("Failed to register sync point: " + sp + ", reason: " + reason);
	}

	@Override
	public void synchronizationPointRegistrationSucceeded(String label) {
		SynchronizationPoint sp = SynchronizationPoint.lookup(label);
		if(sp != null){
			sp.isRegistered(true);
			System.out.println("Successfully registered sync point: " + sp);
		}
		else
			throw new IllegalArgumentException("SynchronizationPoint["+sp+"] not
defined.");
	}

	@Override
	public void announceSynchronizationPoint(String label, byte[] tag) {
		SynchronizationPoint sp = SynchronizationPoint.lookup(label);
		if(sp != null){
			sp.isAnnounced(true);
			logger.log(Level.INFO, "Synchronization point announced: " + sp);
		}
		else
			throw new IllegalArgumentException("SynchronizationPoint["+sp+"] not
defined.");
	}

	@Override
	public void federationSynchronized(String label, FederateHandleSet failed) {
		SynchronizationPoint sp = SynchronizationPoint.lookup(label);
		if(sp != null){
			sp.federationIsSynchronized(true);
			System.out.println("Federation Synchronized: " + sp);
		}
		else
			throw new IllegalArgumentException("SynchronizationPoint["+sp+"] not defined");
	}

}
```

The HLAModule class

```
package federate;

import skf.core.*;
import synchronizationPoint.SynchronizationPoint;
```

```
import hla.rti1516e.RTIambassador;
import hla.rti1516e.exceptions.*;

public class HLAModule {

	private SEEAbstractFederateAmbassador fed_amb = null;
	private RTIambassador rti_amb = null;

	public HLAModule(SEEAbstractFederateAmbassador fed_amb) throws RTIinternalError {
		this.fed_amb = fed_amb;
		this.rti_amb  = SEERTIAmbassador.getInstance();
	}

	// ********************** SynchronizationPoint ********************
	public void achieveSynchronizationPoint(SynchronizationPoint sp) {
		try {
			rti_amb.synchronizationPointAchieved(sp.getValue());
		} catch (SynchronizationPointLabelNotAnnounced | SaveInProgress
				| RestoreInProgress | FederateNotExecutionMember | NotConnected
				| RTIinternalError e) {
			e.printStackTrace();
		}
	}

	public void registerSynchronizationPoint(SynchronizationPoint sp) {
		try {
			rti_amb.registerFederationSynchronizationPoint(sp.getValue(), null);
		} catch (SaveInProgress | RestoreInProgress
				| FederateNotExecutionMember | NotConnected | RTIinternalError e) {
			e.printStackTrace();
		}
	}

	public void announceSynchronizationPoint(SynchronizationPoint sp) {
		try {
			fed_amb.announceSynchronizationPoint(sp.getValue(), null);
		} catch (FederateInternalError e) {
			e.printStackTrace();
		}
	}

}
```

The ShutdownTask class

```
package federate;

import hla.rti1516e.exceptions.*;
import synchronizationPoint.SynchronizationPoint;
```

```
public class ShutdownTask implements Runnable {

	private TestFederate federate = null;

	public ShutdownTask(TestFederate federate) {
		this.federate  = federate;
	}

	@Override
	public void run() {
		try {
		waitingForAnnouncement(SynchronizationPoint.MTR_SHUTDOWN);


	federate.getHLAModule().achieveSynchronizationPoint(SynchronizationPoint.MTR_SHUTDOWN);

		waitingForSynchronization(SynchronizationPoint.MTR_SHUTDOWN);
		} catch (InterruptedException e) {e.printStackTrace(); }
		// 5. Disconnect The Federate from the RTI
		try {
			federate.diconnectFromRTI();
		} catch (Exception e) { e.printStackTrace();}
	}

	private void waitingForAnnouncement(SynchronizationPoint sp) throws InterruptedException {
		while(!sp.isAnnounced())
			Thread.sleep(10);
	}

	private void waitingForSynchronization(SynchronizationPoint sp) throws InterruptedException {
		while(!sp.federationIsSynchronized())
			Thread.sleep(10);
	}
}
```

The ExecutionConfiguration class (ObjectClass)

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

import skf.coder.HLAfloat64LECoder;
import skf.coder.HLAunicodeStringCoder;
import skf.model.object.annotations.Attribute;
import skf.model.object.annotations.ObjectClass;

@ObjectClass(name = "ExecutionConfiguration") (note [2])
public class ExecutionConfiguration {
```

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

@Attribute(name = "scenario_time_epoch", coder = HLAfloat64LECoder.class)
private Double scenario_time_epoch = null;

@Attribute(name = "next_mode_scenario_time", coder = HLAfloat64LECoder.class)
private Double next_mode_scenario_time = null;

@Attribute(name = "next_mode_cte_time", coder = HLAfloat64LECoder.class)
private Double next_mode_cte_time = null;

@Attribute(name = "current_execution_mode", coder = ExecutionModeCoder.class)
private ExecutionMode current_execution_mode = null;

@Attribute(name = "next_execution_mode", coder = ExecutionModeCoder.class)
private ExecutionMode next_execution_mode = null;

public ExecutionConfiguration() {}

public ExecutionConfiguration(String root_frame_name,
                              Double scenario_time_epoch, Double next_mode_scenario_time,
                              Double next_mode_cte_time, ExecutionMode current_execution_mode,
                              ExecutionMode next_execution_mode) {

        this.root_frame_name = root_frame_name;
        this.scenario_time_epoch = scenario_time_epoch;
        this.next_mode_scenario_time = next_mode_scenario_time;
        this.next_mode_cte_time = next_mode_cte_time;
        this.current_execution_mode = current_execution_mode;
        this.next_execution_mode = next_execution_mode;

}

/**
* @return the root_frame_name
*/
public String getRoot_frame_name() {
        return root_frame_name;
}

/**
* @param root_frame_name the root_frame_name to set
*/
public void setRoot_frame_name(String root_frame_name) {
        this.root_frame_name = root_frame_name;
}

/**
* @return the scenario_time_epoch
*/
```

```
public Double getScenario_time_epoch() {
        return scenario_time_epoch;
}

/**
* @param scenario_time_epoch the scenario_time_epoch to set
*/
public void setScenario_time_epoch(Double scenario_time_epoch) {
        this.scenario_time_epoch = scenario_time_epoch;
}

/**
* @return the next_mode_scenario_time
*/
public Double getNext_mode_scenario_time() {
        return next_mode_scenario_time;
}

/**
* @param next_mode_scenario_time the next_mode_scenario_time to set
*/
public void setNext_mode_scenario_time(Double next_mode_scenario_time) {
        this.next_mode_scenario_time = next_mode_scenario_time;
}

/**
* @return the next_mode_cte_time
*/
public Double getNext_mode_cte_time() {
        return next_mode_cte_time;
}

/**
* @param next_mode_cte_time the next_mode_cte_time to set
*/
public void setNext_mode_cte_time(Double next_mode_cte_time) {
        this.next_mode_cte_time = next_mode_cte_time;
}

/**
* @return the current_execution_mode
*/
public ExecutionMode getCurrent_execution_mode() {
        return current_execution_mode;
}

/**
* @param current_execution_mode the current_execution_mode to set
*/
public void setCurrent_execution_mode(ExecutionMode current_execution_mode) {
        this.current_execution_mode = current_execution_mode;
```

```
	}

	/**
	* @return the next_execution_mode
	*/
	public ExecutionMode getNext_execution_mode() {
		return next_execution_mode;
	}

	/**
	* @param next_execution_mode the next_execution_mode to set
	*/
	public void setNext_execution_mode(ExecutionMode next_execution_mode) {
		this.next_execution_mode = next_execution_mode;
	}

	/* (non-Javadoc)
	* @see java.lang.Object#toString()
	*/
	@Override
	public String toString() {
		return "ExecutionConfiguration [root_frame_name=" + root_frame_name
				+ ", scenario_time_epoch=" + scenario_time_epoch
				+ ", next_mode_scenario_time=" + next_mode_scenario_time
				+ ", next_mode_cte_time=" + next_mode_cte_time
				+ ", current_execution_mode=" + current_execution_mode
				+ ", next_execution_mode=" + next_execution_mode + "]";
	}
}
```

The ExecutionMode class

```
package model.objectClass;

public enum ExecutionMode {

	EXEC_MODE_UNINITIALIZED((short)0),
	EXEC_MODE_INITIALIZING((short)1),
	EXEC_MODE_RUNNING((short)2),
	EXEC_MODE_FREEZE((short)3),
	EXEC_MODE_SHUTDOWN((short)4);

	private short value;

	private ExecutionMode(short value){
		this.value = value;
	}
```

```
	public static ExecutionMode lookup(short value) {
		for(ExecutionMode rm : ExecutionMode.values())
			if(rm.value == value)
				return rm;
		return null;
	}

	public short getValue() {
		return this.value;
	}
}
```

The ExecutionModeCoder class

```
package model.objectClass;

import hla.rti1516e.RtiFactoryFactory;
import hla.rti1516e.encoding.DecoderException;
import hla.rti1516e.encoding.EncoderFactory;
import hla.rti1516e.encoding.HLAinteger16LE;
import hla.rti1516e.exceptions.RTIinternalError;
import skf.coder.Coder;

public class ExecutionModeCoder implements Coder<ExecutionMode> {

	private EncoderFactory factory = null;
	private HLAinteger16LE value = null;

	public ExecutionModeCoder() throws RTIinternalError {
		this.factory = RtiFactoryFactory.getRtiFactory().getEncoderFactory();
		this.value = factory.createHLAinteger16LE();

	}

	@Override
	public ExecutionMode decode(byte[] arg0) throws DecoderException {
		value.decode(arg0);
		return ExecutionMode.lookup(value.getValue());

	}

	@Override
	public byte[] encode(ExecutionMode arg0) {
		value.setValue(arg0.getValue());
		return value.toByteArray();
	}
```

```
	@Override
	public Class<ExecutionMode> getAllowedType() {
		return ExecutionMode.class;
	}
}
```

The ModeTransitionRequest class (InteractionClass)

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

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

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

	@Parameter(name = "execution_mode", coder = MTRModeCoder.class)
	private MTRMode execution_mode = null;

	public ModeTransitionRequest(){}

	public ModeTransitionRequest(MTRMode execution_mode) {
		this.execution_mode = execution_mode;
	}

	/**
	* @return the execution_mode
	*/
	public MTRMode getExecution_mode() {
		return execution_mode;
	}

	/**
	* @param execution_mode the execution_mode to set
	*/
	public void setExecution_mode(MTRMode execution_mode) {
		this.execution_mode = execution_mode;
	}

	/* (non-Javadoc)
	* @see java.lang.Object#hashCode()
	*/
	@Override
	public int hashCode() {
		final int prime = 31;
```

```
		int result = 1;
		result = prime * result
				+ ((execution_mode == null) ? 0 : execution_mode.hashCode());
		return result;
	}

	/* (non-Javadoc)
	* @see java.lang.Object#equals(java.lang.Object)
	*/
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		ModeTransitionRequest other = (ModeTransitionRequest) obj;
		if (execution_mode != other.execution_mode)
			return false;
		return true;
	}

	/* (non-Javadoc)
	* @see java.lang.Object#toString()
	*/
	@Override
	public String toString() {
		return "ModeTransitionRequest [execution_mode=" + execution_mode + "]";
	}

}
```

The MTRMode class

```
package model.interactionClass;

public enum MTRMode {

	MTR_GOTO_RUN((short)2),
	MTR_GOTO_FREEZE((short)3),
	MTR_GOTO_SHUTDOWN((short)4);

	private short value;

	private MTRMode(short value){
```

```
                this.value = value;
        }

        public short getValue(){
                return value;
        }

        public static MTRMode lookup(short value){
                for(MTRMode mode : MTRMode.values())
                        if(mode.value == value)
                                return mode;
                return null;
        }
}
```

The MTRModeCoder class

```
package model.interactionClass;

import hla.rti1516e.RtiFactoryFactory;
import hla.rti1516e.encoding.DecoderException;
import hla.rti1516e.encoding.EncoderFactory;
import hla.rti1516e.encoding.HLAinteger16LE;
import hla.rti1516e.exceptions.RTIinternalError;
import skf.coder.Coder;

public class MTRModeCoder implements Coder<MTRMode> {

        private HLAinteger16LE coder = null;
        private EncoderFactory factory = null;

        public MTRModeCoder() throws RTIinternalError {
                this.factory = RtiFactoryFactory.getRtiFactory().getEncoderFactory();
                this.coder = factory.createHLAinteger16LE();
        }

        @Override
        public MTRMode decode(byte[] code) throws DecoderException {
                coder.decode(code);
                return MTRMode.lookup(coder.getValue());
        }

        @Override
        public byte[] encode(MTRMode element) {
                coder.setValue(element.getValue());
                return coder.toByteArray();
```

```
	}

	@Override
	public Class<MTRMode> getAllowedType() {
		return MTRMode.class;
	}

}
```

The SyncronizationPoint class

```
package synchronizationPoint;

public enum SynchronizationPoint {

	INITIALIZATION_STARTED("initialization_started"),
	INITIALIZATION_COMPLETED("initialization_completed"),
	OBJECTS_DISCOVERED("objects_discovered"),
	MTR_RUN("mtr_run"),
	MTR_FREEZE("mtr_freeze"),
	MTR_SHUTDOWN("mtr_shutdown");

	private String value = null;
	private boolean isAnnounced = false;
	private boolean isRegistered = false;
	private boolean federationIsSynchronized = false;

	private SynchronizationPoint(String value) {
		this.value = value;
	}

	public static SynchronizationPoint lookup(String value) {
		for(SynchronizationPoint sp : SynchronizationPoint.values())
			if(sp.value.equalsIgnoreCase(value))
				return sp;
		return null;
	}

	public String getValue() {
		return this.value;
	}

	public void isAnnounced(boolean value) {
		this.isAnnounced = value;
	}
```

```
        public void federationIsSynchronized(boolean value) {
                this.federationIsSynchronized = value;
        }

        public boolean isAnnounced() {
                return isAnnounced;
        }

        public boolean federationIsSynchronized() {
                return federationIsSynchronized;
        }

        public void isRegistered(boolean value) {
                this.isRegistered = value;

        }

        public boolean isRegistered() {
                return isRegistered;

        }

}
```

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.

[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.

The Configuration file

```
{
 "asynchronousDelivery" : true,
 "crcHost" : "localhost",
 "crcPort" : 8989,
 "federateName" : "SEETestFederate",
 "federateType" : "Test",
 "federationName" : "SEE 2017",
 "fomDirectory" : "C: \\foms",
 "realtime" : false,
 "simulationEphoc" : "2015-04-19T20:00:00.000Z", (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 2017 event, the Simulation Ephoc has been set to 04/19/2015 20:00:00 GMT.