A JAVA LIBRARY FOR EASING THE DISTRIBUTED SIMULATION OF SPACE SYSTEMS

Alberto Falcone, Alfredo Garro

Department of Informatics, Modeling, Electronics, and Systems Engineering (DIMES), University of Calabria, via P. Bucci 41C, 87036, Rende (CS), Italy

{alberto.falcone, alfredo.garro}@dimes.unical.it

ABSTRACT

The space flight domain is one of the numerous fields that involve experts belonging to different scientific domains such as mathematical, physical, aerospace and software engineering. Many research efforts are focusing on the definition of methods, tools and software libraries, mainly aiming at providing a robust and flexible way for defining, building and simulating complex systems in space so as to understand, predict and optimize their behavior. In this context, the paper presents a space flight dynamics library, named *Java Space Dynamics Library (JSDL)*, which offers high fidelity models and algorithms to manage space systems according to the SISO Space Reference FOM standardization initiative.

Keywords: Modeling and Simulation, Space Flight Dynamics, Distributed Simulations, High Level Architecture (HLA)

1. INTRODUCTION

Due to the increasing complexity of space systems, and thus of the related engineering problems (Falcone, Garro, and Tundis 2014; Fortino et al. 2007; Garro et al. 2015; Garro and Falcone 2015), there is a consistent investment in the development of new methods, tools and software libraries able to provide a robust and flexible way for defining, building and simulating them (Falcone et al. 2016; Fortino et al. 2006; Ido 2012; Rogovchenko-Buffoni et al. 2014; San-Juan et al. 2011). These available, commercial and noncommercial, solutions support one or more of the phases in the development of space systems such as flight mechanics, propulsion, orbit controls and data analysis; however, none of them seems capable of providing complete coverage of the whole development process in a flexible way (Pulecchi and Lovera 2006).

In this context, there is an increasing need for efficient and flexible solutions capable of covering all the steps in the design and develop of space systems, especially for supporting system modeling and simulation where modularity, flexibility and reusability are key features to provide (Falcone et al. 2016; Falcone et al. 2015; Pulecchi and Lovera 2006).

To contribute to fill this lack, the paper presents the *Java Space Dynamics Library (JSDL)* project, emphasizing its flexibility and showing the set of services provided to define and build space systems such as satellites and spacecrafts. The rest of the paper is structured as follows: related works are discussed in Section 2; Section 3 presents the *Java Space Dynamics Library (JSDL)* whose architecture and provided services are discussed in Section 4 and 5 respectively. Finally, in Section 6 conclusions are drawn and future research directions are delineated.

2. RELATED WORK

There are several research efforts on the development of methods, tools and libraries in the astrodynamics field, mainly aiming at providing a robust and flexible way for defining, building and simulating complex systems in space. The most applicable solutions have been developed after the mid-1960's when space missions were the attention of media and computers become prevalent in academia and industry.

The *Java Astrodynamics Toolkit (JAT)* is an open source library of reusable components, distributed under the GNU General Public License (GLP). It is implemented in the Java language and helps developers to create their own application programs and solve problems in astrodynamics, mission design, spacecraft navigation, guidance and control. It provides functionalities that allow the rapid development of spacecraft simulations including 2D and 3D visualization capabilities. Possible applications of JAT include: (i) Design and analysis of space missions, including trajectory optimization; (ii) Simulation of spacecraft navigation, guidance and control as well as its visualization in a 3D environment; and, (iii) Simulation of the motion for basic rigid and flexible spacecraft dynamics (Gaylor, Page, and Bradley 2006).

Another software library that enables developers to effectively define and manage elements in space is *Orbits Extrapolation Kit (Orekit)* (CS Communication & Systémes 2017). *Orekit* is implemented in the Java language and aims at providing accurate and efficient low level standard astrodynamical models (e.g., time, frames, orbital parameters, orbit propagation, attitude

and celestial bodies) and algorithms (e.g., time conversions, propagations and pointing) for the development of flight dynamics applications. It is designed to be easily used in very different contexts, from quick studies up to critical operations. It was developed in 2002 at CS Systémes d'Information and was officially released as an open source software, under the Apache License Version 2.0, in 2008 (CS Communication & Systémes 2017).

European Space Agency (ESA) engineers have been developing several spacecraft simulation tools that form the *Mission - Customer Furnished Item (CFI) Software (Mission CFI)*. It includes the following products (ESA 2017):

- The *Earth Observation CFI (EOCFI)* software, which is a collection of multiplatform precompiled C libraries for timing, coordinate conversions, orbit propagation, satellite pointing calculations, and target visibility calculations, specifically parametrized and configured for EO satellites;
- The *EO Orbit and Attitude Adapter (EO Adapter)*, which is part of the Earth Observation Mission Software Suite. It is a tool/library to generate Orbit and Attitude files compliant with EOCFI format using data extracted from one or more binary files, for example files containing Telemetry packets including Orbit and Attitude information;
- The *Envisat CFI* software, which is a collection of multiplatform precompiled C libraries for timing, coordinate conversions, orbit propagation, satellite pointing calculations, and target visibility calculations, specifically parametrized and configured for the Envisat satellite.

The JSDL project presented in this Section stems from the SISO Space Reference FOM standardization initiative carried out by the SISO Space Reference FOM (SRFOM) Product Development Group (PDG) (Möller et al. 2016). JSDL aims at supporting the development of complex space systems by providing high fidelity models and algorithms to manage them. Differently from proprietary and commercial solutions that require tool-specific knowledge and training, JSDL is an open source project released under the open source policy Lesser GNU Public License (LGPL) and can be freely and easily customized and/or extended to cover specific domain aspects. This license allows anybody to build both commercial and noncommercial applications without restrictions or limitations from the use of JSDL. In the following sections the JSDL project is described in details by highlighting its architecture and functionalities.

3. THE JSDL PROJECT

Java Space Dynamics Library (JSDL) is a low-level space dynamics library that facilitates the design and development of space systems, such as space vehicles and satellites. The open source nature of the library allows developers to investigate and customize the architecture and functionalities defined in the source code to fit their own needs.

The JSDL has been designed and developed in the context of the research activities carried out within the SMASH-Lab (System Modeling And Simulation Hub - Laboratory) of the University of Calabria (Italy) working in cooperation with the SISO Space Reference FOM (SRFOM) Product Development Group (PDG) (Möller et al. 2016). The primary goal of JSDL is to provide high fidelity models and algorithms needed for defining space systems that are as accurate and robust as those provided by existing commercial and government software. It is fully implemented in the Java programming language and provides a consistent set of functionalities for developing and running complex elements in space such as, time scales, reference frames, orbital parameters, orbit propagation, and attitude.

The JSDL provides to developers the following resources: (i) the *technical documentation* that describes the library with its philosophy and mission; (ii) the *user guide* to support developers in the use of the library; and (iii) a set of *reference examples* that show how to create space systems.

In the following, the attention is focused on the architecture and services provided by the library.

4. ARCHITECTURE OF THE JSDL

The JSDL library depends only on the Java Standard Edition version 7 (or above), Apache Commons Math (Apache Commons 2017) version 3.6 and JDateTime (Jodd Components 2017) version 3.8 libraries at runtime. The JSDL provides a set of services, each of which defines some Java classes and interfaces that enable specific functionalities. The JSDL architecture is shown in Figure 1.

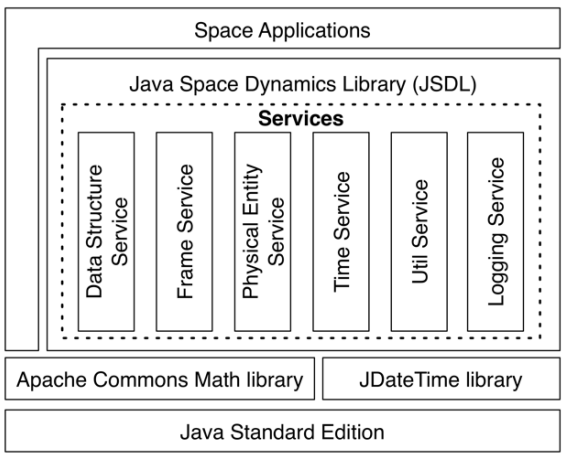

Figure 1: Architecture of the JSDL library.

Space Applications. Contains the space applications that are built using the functionalities provided by the JSDL. An application can interact with the Apache Commons Math and JDateTime directly or through the JSDL library.

Java Space Dynamics Library (JSDL). It is the core library for creating Space applications. It provides a set of features useful for modeling objects in space. The complexity of the features provided is hidden behind an intuitive set of APIs.
Apache Commons Math library. It is a standard library of lightweight, self-contained mathematics and statistics components addressing the most common practical problems not immediately available in the Java programming language (Apache Commons 2017).
JDateTime library. It is a library that offers a very precise way to track dates and time. It uses well-defined and proven astronomical algorithms for time manipulation (Jodd Components 2017).
In the following Sections, the six JSDL services with their UML Class diagrams are described in detail.

5. SERVICES OF THE JSDL

5.1. Data Structure Service

The *Data Structure Service* defines functionalities that ease working with complex data structures. It provides a very useful set of data structures (tree and queue) to build and manage *Reference Frames* and *Physical Entities* with their transformations.
The structure of the *Data Structure Service* is shown in Figure 2 by using a UML Class Diagram.

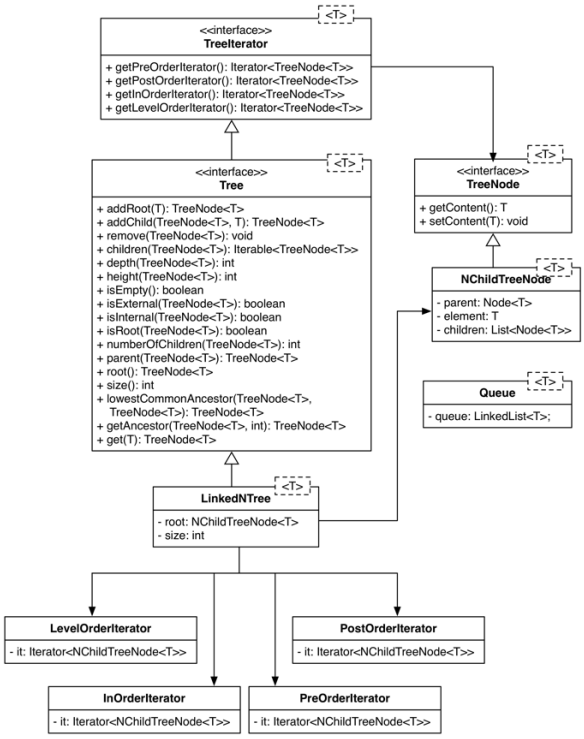

Figure 2: The architecture of the Data Structure Service.

The *LinkedNTree* is a generic class that stores elements hierarchically where each element has a parent element and zero or more children elements. It implements the Tree interface that defines some functionalities to handle a tree such as *height(), depth(), root()* and *size()*. Moreover, all the common traversal schemes for trees are provided: *LevelOrderIterator, PreOrderIterator, InOrderIterator* and *PostOrderIterator*.
The *Queue* class provides a queue data structure that follows the First-in First-out (FIFO) strategy. Elements can only be added to the end (enqueue) and only be removed from the front (dequeue). The queue has been implemented by using a Java standard *LinkedList* and provides two methods *enqueue()* and *dequeue()* to perform each task respectively.

5.2. Frame Service

Reference frame is a fundamental concept for representing when and where a physical entity exists in time and space (Falcone et al. 2014; Möller et al. 2016). This representation is referred to as the state of the entity. In order to represent the state of something, it is necessary to express that state with respect to some time scale and some referent coordinate system. This combination of time and coordinate system is referred as a *Space-Time Coordinate* or *Reference Frame* (Möller et al. 2016). The structure of the Frame Service is shown in Figure 3 by using a UML Class Diagram.

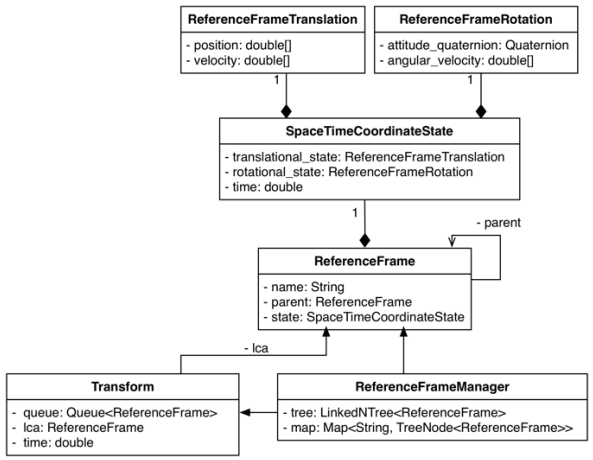

Figure 3: The architecture of the Frame Service.

The *Frame Service* provides functionalities to handle Reference Frames. It includes the fundamental *ReferenceFrame* class that represents a single frame. Each Reference Frame, as defined in the SISO Space Reference FOM (Möller et al. 2016), is composed of three attributes: (i) *name*, which represents the unique name of the reference frame; (ii) *parent,* which is the parent Reference Frame. If it is *NULL*, the Reference Frame is the root frame; and (iii) *space-time coordinate state*, which defines through the *SpaceTimeCoordinateState* Class a four-dimensional representation of the space-time coordinate state with respect to its parent reference frame (Möller et al. 2016). It consists of:

- *Translational state information*, which provides through the *ReferenceFrameTranslation* class a position vector $\vec{v}$ from the origin of the parent reference frame to the origin of the reference frame. It

also provides a velocity vector $\vec{v}$ for the motion of the reference frame with respect to the parent frame. Both of these vectors are expressed with respect to the parent reference frame. These vectors can be used to describe the translational position and motion of a frame with respect to its parent;

- *Rotational state information*, which provides through the *ReferenceFrameRotation* class an attitude quaternion $\tilde{q}$ that describes the attitude of the reference frame with respect to its parent frame. It also provides an angular velocity vector $\vec{w}$ that describes the rotational motion of the reference frame with respect to the parent frame expressed in the subject frame's coordinates. $\tilde{q}$ and $\vec{w}$ can be used to describe the attitude and rotational motion of a frame with respect to its parent.
- *Time,* which contains information about the time t to which the *space-time coordinate state* corresponds.

As shown in Figure 4, all Reference Frames are organized as a tree that is formed from a single base root node with directed paths from an arbitrary number of child nodes.

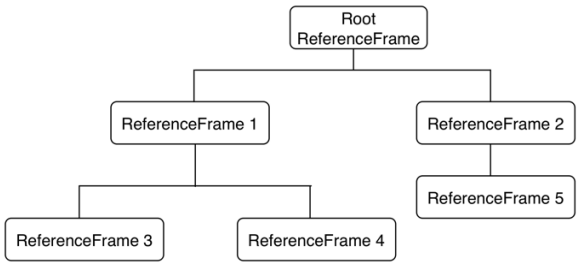

Figure 4: Tree of ReferenceFrames.

These child nodes can then have directed paths from other arbitrary sets of child nodes.

The translational and rotational information can be used to transform a generic vector expressed in a given reference frame $\vec{r}_{child}$ into a vector expressed in its parent frame $\vec{r}_{parent}$. In turn, the vector $\vec{r}_{parent}$ now expressed in the parent frame can be expressed in the parent's parent frame or in another child frame of the parent frame. Chaining together sequences of transformations using the relationships established in the reference frame tree allows for transformation between any pair of frames in the reference frame tree.

Transformations are defined and managed by the *Transform* and *ReferenceFrameManager* classes. In particular, a transformation is computed by merging individual transforms while walking the shortest path between them. The walking/merging operations are handled transparently by the library. Developers only need to select the frames, provide the date and ask for the transformation, without knowing how the frames are related to each other. Transformations are defined as operators that when applied to the coordinates of a vector expressed in the *initial Reference Frame*, provide the coordinates of the same vector expressed in the *final Reference Frame*.

Equation 1 gives the transformation of a position vector expressed in a child reference frame into a position vector expressed in the parent reference frame (Kuipers 2002),

$$\vec{r}_{parent} = \vec{r}_{0\_parent} + \tilde{Q}(\vec{r}_{child}) \quad (1)$$

where $\vec{r}_{child}$ is the position vector expressed in the child reference frame, $\tilde{Q}(\vec{r}_{child})$ is the quaternion rotation operator associated with the attitude quaternion $\tilde{q}$ that defines the attitude of the child reference frame with respect to the parent reference frame; $\vec{r}_{0\_parent}$ is the vector giving the position of child reference frame origin with respect to the parent reference frame origin expressed in parent reference frame coordinates; $\vec{r}_{parent}$ is the position vector of the entity expressed in parent reference frame coordinates.

With reference to the $\tilde{Q}(\vec{r}_{child})$ operation, it is the canonical way of multiplying a quaternion $\tilde{q}$ by a vector $\vec{x}$ as given by expression (2),

$$\tilde{Q}(\vec{x}) = \tilde{q} \cdot \vec{x} \cdot \tilde{q}^* \quad (2)$$

where $\tilde{q}^*$ is the conjugate of $\tilde{q}$.

The relative motion between a child reference frame and a parent reference frame is provided by the velocity $\vec{v}$ and angular velocity $\vec{w}$ vectors. Equation 3 gives the velocity of an entity expressed in the parent reference frame given the velocity of the entity expressed in the child reference frame (Kuipers 2002),

$$\vec{v}_{parent} = \vec{v}_{0\_parent} + \tilde{Q}(\vec{v}_{child} + (\vec{w}_{child} \times \vec{r}_{child})) \quad (3)$$

where $\vec{v}_{child}$ is the velocity vector of an entity expressed in the child reference frame, $\vec{w}_{child}$ is the angular velocity vector of the child frame with respect to the parent frame and expressed in child frame coordinates, $\vec{v}_{o\_parent}$ is the velocity of the child frame with respect to the parent frame expressed in parent frame coordinates, and $\vec{v}_{parent}$ is the velocity of an entity expressed the parent reference frame.

In most cases, the position and velocity relationships are sufficient. However, acceleration is sometimes needed and is included for completeness. Equation 4 gives the acceleration of an entity expressed in the parent reference frame given the acceleration of the entity expressed in the child reference frame (Kuipers 2002),

$$\vec{a}_{parent} = \vec{a}_{0\_parent} + \tilde{Q}\left(\vec{a}_{child} + \left(\vec{w}_{child} \times (\vec{w}_{child} \times \vec{r}_{child})\right) + (2\vec{w}_{child} \times \vec{v}_{child}) + (\vec{\alpha}_{child} \times \vec{r}_{child})\right) \quad (4)$$

where $\vec{a}_{child}$ is the acceleration of an entity expressed in the child reference frame, $\vec{\alpha}_{child}$ is the angular

acceleration of the child frame with respect to the parent frame and expressed in child frame coordinates, $\vec{a}_{0\_parent}$ is the acceleration of the child frame with respect to the parent frame expressed in parent frame coordinates, and $\vec{a}_{parent}$ is the acceleration of an entity expressed in the parent reference frame.
Concerning reverse transformations; using the child to parent vector transformation equations above defined along with some vector and quaternion algebra, the resulting equation 5 gives the transformation of a position vector expressed in a parent reference frame into a position vector expressed in the child reference frame (Kuipers 2002),

$$\vec{r}_{child} = \tilde{Q}^*\left(\vec{r}_{parent} - \vec{r}_{0\_parent}\right) = -\vec{r}_{0\_child} + \tilde{Q}^*\left(\vec{r}_{parent}\right) \quad (5)$$

where $\tilde{Q}^*\left(\vec{r}_{parent}\right)$ is the conjugate quaternion rotation operator associated with the attitude quaternion $\tilde{q}$ that defines the attitude of the child reference frame with respect to the parent reference frame, and $\vec{r}_{0\_child}$ is the vector giving the position of child reference frame origin with respect to the parent reference frame origin expressed in child reference frame coordinates (Kuipers 2002).

$$\vec{v}_{child} = \tilde{Q}^*\left(\vec{v}_{parent} - \vec{v}_{0\_parent}\right) - \left(\vec{w}_{child} \times \vec{r}_{child}\right) = -\vec{v}_{0\_child} - \left(\vec{w}_{child} \times \vec{r}_{child}\right) + \tilde{Q}^*\left(\vec{v}_{parent}\right) \quad (6)$$

Similar relationships can be derived for velocity (Equation 6) and acceleration (Equation 7) (Kuipers 2002).

$$\vec{a}_{child} = \tilde{Q}^*\left(\vec{a}_{parent} - \vec{a}_{0\_parent}\right) - \left(\vec{w}_{child} \times \left(\vec{w}_{child} \times \vec{r}_{child}\right)\right) - \left(2\vec{w}_{child} \times \vec{v}_{child}\right) - \left(\vec{\alpha}_{child} \times \vec{r}_{child}\right) = -\vec{a}_{0\_child} - \left(\vec{w}_{child} \times \left(\vec{w}_{child} \times \vec{r}_{child}\right)\right) - \left(2\vec{w}_{child} \times \vec{v}_{child}\right) - \left(\vec{\alpha}_{child} \times \vec{r}_{child}\right) + \tilde{Q}^*\left(\vec{a}_{parent}\right) \quad (7)$$

5.3. Physical Entity Service

The structure of the Physical Entity Service is shown in Figure 5 by using a UML Class Diagram.
PhysicalEntity is the highest-level object class in the JSDL entity hierarchy. This class provides attributes to describe an entity's location in time and space. It also contains attributes to uniquely identify it individually from all other physical entities.
Physical entities have two intrinsically associated reference frames: (i) a *structural frame*; and (ii) a *body frame*. These are not registered in the reference frame tree but are used to place and orient the entity in space with respect to a reference frame in the tree. The origin of the *structural frame* is located at some arbitrary but known point on the entity (Möller et al. 2016). The *body frame* origin is at the entity's *center of mass* and is located with respect to the entity's structural reference frame by a vector from the origin of the structural reference frame to the center of mass of the entity. This vector is expressed in the entity's structural reference frame. The orientation of the entity's *body frame* with respect to the entity's structural reference frame is defined by an attitude quaternion.

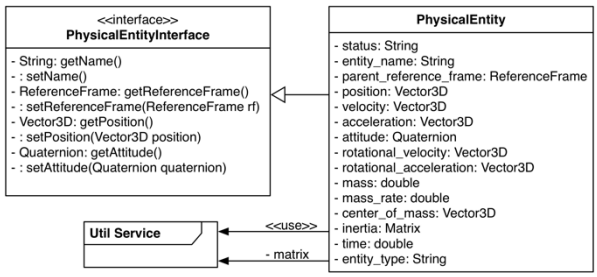

Figure 5: The architecture of the Physical Entity Service.

The *Physical Entity Service* is designed to provide functionalities for space objects such as satellites, asteroids and vehicles. The core attributes defined in the *PhysicalEntity* class includes the position and orientation with respect to a defined parent reference frame, which must be a reference frame instance in the reference frame tree, and a time tag in a defined time scale. This information is sufficient to unambiguously represent an entity in time and space.

5.4. Time Service

The *Time Service* allows to manage epochs, time scales, time units and to compare time instants. The structure of the *Time Service* is shown in Figure 6 by using a UML Class Diagram.

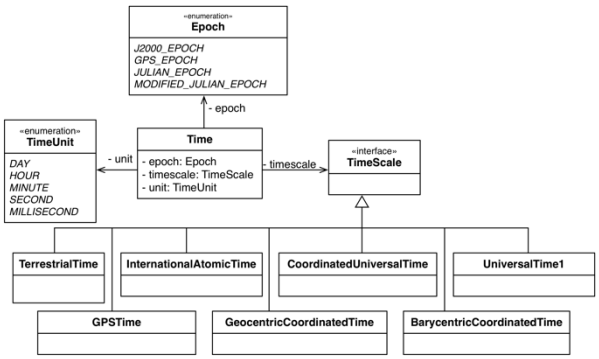

Figure 6: The architecture of the Time Service.

The principal class is *Time* that represents a unique instant in time defined by specifying a point in a specific epoch (e.g., *J2000, GPS* and *Julian epoch*), time scale and time unit (Möller et al. 2016). The *TimeScale* interface defines a set of predefined time scales:

- *Universal Time (UT)*. It is a time standard based on Earth's rotation, defined as the Mean Solar Time at the Royal Observatory in Greenwich, England. There are three variations of Universal Time. UT0 is the observed mean solar time. UT1 is UT0 corrected for polar

motion, the motion of the Earth's rotational axis over the surface of the Earth, and UT2 that is corrected for seasonal variations but today it is considered obsolete.

- *International Atomic Time (TAI)*. It was introduced in 1972 and represents a high-precision atomic coordinate time standard based on the notional passage of proper time on Earth's geoid (Guinot 1986). This time scale is accurate enough to observe relativistic effects for clocks in motion or accelerated by a local gravity field. One advantage of using TAI is that it is a continuous uniform time scale. Specifically, the rate of time passage for TAI is constant unlike the Earth rotation based scales. This means that the Earth rotation based time scales diverge from TAI over time due to the variations in the Earth's rotation. TAI is exactly 36 seconds ahead of UTC. The 36 seconds results from the initial difference of 10 seconds at the start of 1972, plus 26 leap seconds in UTC since 1972.
- *Coordinated Universal Time (UTC)*. It is a 24-hour time standard that is used to synchronize world clocks. UTC is defined by the International Telecommunications Union Recommendation (ITU-R TF.460-6), Standard-frequency and time-signal emissions (Recommendation I., 460-6 2002) and is based on International Atomic Time (TAI) with leap seconds added at irregular intervals to compensate for the slowing of Earth's rotation. Leap seconds are inserted as necessary to keep UTC within 0.9 seconds of universal time, UT1 (Department T.S., United States Naval Observatory).
- *Global Positioning System Time (GPS Time)*. GPS Time is the uniform time scale with a starting epoch at midnight between Saturday January 5th and Sunday January 6th, 1980 (1980 January 6, 00:00:00 UTC). GPS Time counts in weeks and seconds of a week from this instant. The GPS week begins at the transition between Saturday and Sunday. The days of the week are numbered sequentially, with Sunday being 0, Monday 1, Tuesday 2, etc. The GPS time scale begins at the GPS starting epoch with GPS week 0. Within each week, the time is usually denoted as the second of the week (SOW). This is a number between 0 and 604,800 (60 x 60 x 24 x 7). Sometimes SOW is split into a day of week (DOW) between 0 and 6 and a second of day (SOD) between 0 and 86400. While GPST is a uniform time scale, it does have rollover. To limit the size of the numbers used in the data and calculations, the GPS Week Number is a ten-bit count in the range 0-1023, repeating every 1024 weeks. As a result, the week number 'rolled over' from 1023 to 0 at 23:59:47 UTC on Saturday, 21st August 1999. This was before midnight UTC because every GPS week contains exactly 604,800 seconds, to keep the calculations consistent. The 13 intervening leap seconds had put UTC behind GPS system time. The next GPS week rollover occurs on April 6th, 2019.
- *Terrestrial Time (TT)*. It is an astronomical time standard defined by the International Astronomical Union (IAU) used widely for geocentric and topocentric ephemerides. TT is defined to run at the same rate as TAI seconds but with an offset of 32.184 seconds. This offset is based on preserving continuity with other historical dynamic time scales.
- *Geocentric Coordinated Time (TCG)*. It is a coordinate time standard defined in 1991 by the International Astronomical Union (IAU). It is primarily used for theoretical developments based on the Geocentric Celestial Reference System (GCRS). TCG is a relativistic time scale and since the reference frame for TCG is not rotating with the surface of the Earth and not in the gravitational potential of the Earth, TCG ticks faster than clocks on the surface of the Earth by a factor of $6.97 \cdot 10^{-10}$ seconds. TCG, Barycentric Coordinated Time (TCB) and Terrestrial Time (TT) are de- fined in a way that they have the same value on January 1st 1977, 00:00:00 TAI (JD 2443144.5 TAI).
- *Barycentric Coordinated Time (TCB)*. It is a time scale, defined in 1991 by the International Astronomical Union (IAU), primarily used for theoretical developments based on the Barycentric Celestial Reference System (BCRS). TCB is a relativistic time scale and since the reference frame for TCB is not influenced by the gravitational potential caused by the Solar system, TCB ticks faster than clocks on the surface of the Earth by $1.55 \cdot 10^{-8}$ seconds. TCB, Geocentric Coordinated Time (TCG) and Terrestrial Time (TT) are defined in a way that they have the same value on January 1st 1977, 00:00:00 TAI (JD 2443144.5 TAI).

5.5. Util Service

The *Util Service* defines a number of useful functionalities, primarily transformations ones that are useful for working with *Physical Entities* in space. This service should not be considered merely a utility one that is separate from the rest of JSDL; in fact, JSDL depends directly on several of the classes defined in it. Indeed, it provides services needed to define both *Reference Frame* and *Time* objects with their standard conversions.

The structure of the *Util Service* is shown in Figure 7 by using a UML Class Diagram.

The *Matrix* class represents a mathematical matrix. It provides methods for creating matrices, operating on

ISBN 978-88-97999-91-1; Bruzzone, De Felice, Frydman, Longo, Massei and Solis Eds.

them arithmetically and algebraically, and determining their mathematical properties such as trace, rank, inverse and determinant.

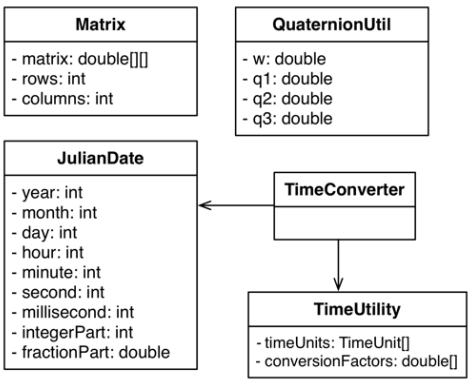

Figure 7: The architecture of the Util Service.

The *QuaternionUtil* class provides classical methods to manage quaternions such as conjugate, inverse and norm. The *JulianDate* class represents a Julian Date, which is a universal time used by all astronomers to ensure that observations are based on a universal astronomical time. It corresponds to the day, hour and minute of the observation and is the interval of time in days since noon at Greenwich on 1 January 4713 BC.
Finally, the *TimeConverter* and *TimeUtility* allow to perform time conversions. Moreover, it is possible to easily convert a *JulianDate* to a standard Java *Calendar* object to have a date/time representation of it through the use of the *toCalendar(JulianDate jd)* method defined in the *TimeConverter* class. For example, the Truncate Julian Date (TJD) 17131.83333333334 can be converted in a Calendar object with value 2015 April 19, 20:00:00 UTC.

5.6. Logging Service

The *Logging Service* provides functionalities useful to both track down any problems or errors occurred during its use, and understand how the JSDL core services work. This information is stored into the *jsdl_trace.log* file.
The structure of the *Logging Service* is shown in Figure 8 through the use of a UML Class Diagram.

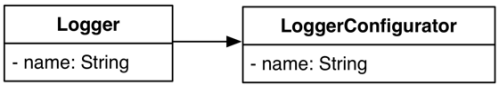

Figure 8: The architecture of the Logging Service.

6. CONCLUSION

In the space flight dynamics domain, many research efforts are focusing on the definition of methods, tools and software libraries, mainly aiming at providing a robust and flexible way for defining, building and simulating complex systems in space.
As discussed in the paper, due to the increasing complexity of space systems and thus of the related engineering problems; new methods, tools and software libraries have been developed in each of these organizations primarily for specific needs and later generalized so as to make them modular, flexible and reusable. The available, commercial and noncommercial, solutions support one or more of the phases in the development of space systems such as flight mechanics, propulsion, orbit controls and data analysis, however none of them seems capable of providing complete coverage of the whole development process of space simulations. To overcome this issue, the *Java Space Dynamics Library (JSDL)* has been created.
The *Java Space Dynamics Library (JSDL)* stems from the SISO Space Reference FOM standardization initiative carried out by the SISO Space Reference FOM (SRFOM) Product Development Group (PDG) (Möller et al. 2016). JSDL is still evolving and aims at supporting the development of complex space systems by providing high fidelity models and algorithms to manage them.

ACKNOWLEDGMENTS

The authors would like to thank all the members of the SISO Space Reference FOM (SRFOM) Product Development Group (PDG) and, in particular, Dr. Edwin Z. Crues (NASA JCS) for their precious advice and suggestions in the development of the Java Space Dynamics Library (JSDL).

REFERENCES

Apache Commons, 2017. Apache Commons Math home page. Available from: https://commons.apache.org/proper/commons-math/ [accessed 20 Feb 2017].

Rogovchenko-Buffoni, L., Tundis, A., Hossain, M.Z., Nyberg, M., and Fritzson, P., 2014. An integrated toolchain for model based functional safety analysis. Journal of Computational Science, 5(3), pp. 408-414.

CS Communication & Systémes, 2017. Orbits Extrapolation Kit (Orekit) home page. Available from: https://www.orekit.org/ [accessed 20 Feb 2017].

Department T.S., United States Naval Observatory. Leap Seconds home page. Retrieved 17 July 2011. Available from: http://tycho.usno.navy.mil/ [accessed 20 Feb 2017].

ESA, 2017. The Mission CFI software home page. Available from: http://eop-cfi.esa.int/index.php/mission-cfi software [accessed 20 Feb 2017].

Falcone, A., Garro, A., and Tundis, A., 2014. Modeling and Simulation for the performance evaluation of the on-board communication system of a metro train. In Proceedings of the 13th International Conference on Modeling and Applied Simulation (MAS 2014), pp. 20-29. September 10-12, Bordeaux (France).

Falcone, A., Garro, A., Longo, F., and Spadafora, F., 2014. Simulation exploration experience: A communication system and a 3D real time

visualization for a moon base simulated scenario. In Proceeding of the 18th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, DS-RT '2014, pp. 113-120, October 1-3, Toulouse, France. IEEE Computer Society.

Falcone, A., Garro, A., Taylor, S. J. E., Anagnostou, A., Chaudhry, N. R., and Salah, O., 2016. Experiences in simplifying distributed simulation: The HLA Development Kit Framework. Journal of Simulation, 1-20. ISSN: 1747-7786, DOI: 10.1057/s41273-016-0039-4. Palgrave Macmillan UK.

Falcone, A., Garro, A., Anagnostou, A., Chaudhry, N. R., Salah, O., and Taylor, S. J. E., 2015. Easing the development of HLA Federates: the HLA Development Kit and its exploitation in the SEE Project. In Proceedings of the 19th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, DS-RT '2015, pp. 50-57. October 14-16, Chengdu, China. IEEE Computer Society.

Fortino, G., Garro, A., Mascillaro, S., and Russo, W., 2007. ELDATool: A Statecharts-based Tool for Prototyping Multi-Agent Systems. In Proceeding of the 8th AI*IA/TABOO Joint Workshop "From Objects to Agents": Agents and Industry: Technological Applications of Software Agents, WOA '2007, pp. 14-19, September 24-25, Genova, Italy.

Fortino, G., Garro, A., Russo, W., Caico, R., Cossentino, M., and Termine, F., 2006. Simulation-driven development of multi-agent systems. In Proceedings of the 4th International Industrial Simulation Conference, ISC '2006, pp. 17-24, June 5-7, Palermo, Italy. EUROSIS.

Gaylor, D., Page, R. and Bradley, K., 2006. Testing of the java astrodynamics toolkit propagator. In AIAA/AAS Astrodynamics Specialist Conference and Exhibit, p. 6754.

Garro, A., and Falcone, A., 2015. On the integration of HLA and FMI for supporting interoperability and reusability in distributed simulation. In Proceedings of the Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium, DEVS 2015, Part of the 2015 Spring Simulation Multi-Conference, SpringSim 2015, pp. 9-16. April 12-15, Alexandria, USA,. Society for Computer Simulation International.

Garro, A., Falcone, A., Chaudhry, N. R., Salah, O. A., Anagnostou, A., and Taylor, S. J., 2015. A prototype HLA development kit: results from the 2015 simulation exploration experience. In Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation, pp. 45-46, June 10-12, London, UK. IEEE Computer Society.

Guinot, B., 1986: Is the international atomic time tai a coordinate time or a proper time? Celestial mechanics 38(2), pp. 155–161.

Ido, H., 2012. Space Flight Dynamics as a Service.

Jodd Components, 2017. JDateTime home page. Available from: http://www.jodd.org/doc/jdatetime.html [accessed 20 Feb 2017]

Möller, B., Garro, A., Falcone, A., Crues, E. Z. and Dexter, D. E., 2016. Promoting a-priori interoperability of HLA-based Simulations in the Space domain: the SISO Space Reference FOM initiative. Proceedings of the 20th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications DS-RT 2016, pp. 100-107. September 21-23, London, UK. IEEE Computer Society.

Kuipers, J., 2002. Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality.

Pulecchi, T. and Lovera, M., 2006. A modelica library for space flight dynamics. In Proceedings of the 5th International Modelica Conference.

San-Juan, J. F., Lara, M., López, R., López, L. M., Folcik, Z. J., Weeden, B. and Cefola, P. J., 2011. Using the DSST semi-analytical orbit propagator package via the NonDyWebTools/ AstroDyWebTools open science environment. RdM, 6(1), 2π.

Recommendation, I.: 460-6, 2002. Standard-frequency and time-signal emissions (questionitu-r 102/7). ITU-R Recommendations: Time Signals and Frequency Standards Emission, Geneva, International Telecommunications Union, Radio-communication Bureau.

AUTHORS BIOGRAPHY

Alberto Falcone

Alberto Falcone is a PhD student in Information and Communication Engineering for Pervasive Intelligent Environments at University of Calabria (Italy). In 2016, he was Visiting Researcher at NASA Johnson Space Center (JSC), working with the Software, Robotics, and Simulation Division (ER). He is a member of the Executive Committee as Student Team Coordinator of the Simulation Exploration Experience (SEE) project.

Alfredo Garro

Alfredo Garro is an Associate Professor of Computer and Systems Engineering at the Department of Informatics, Modeling, Electronics and Systems Engineering (DIMES) of the University of Calabria (Italy). In 2016, he was Visiting Professor at NASA Johnson Space Center (JSC), working with the Software, Robotics, and Simulation Division (ER). He is vice chair of the Space Reference Federation Object Model (SRFOM) Product Development Group (PDG) of SISO. He is the Technical Director of the "Italian Chapter" of INCOSE. He is involved as an IEEE senior member in the activities of the IEEE Computer Society, IEEE Reliability Society and IEEE Aerospace and Electronic Systems Society.