REAL-TIME IMPLEMENTATION OF LOW-COST UNIVERSITY SATELLITE 3-AXIS ATTITUDE DETERMINATION AND CONTROL SYSTEM

Synara Rosa Gomes dos Santos, synararosa@gmail.com
Jacques Waldmann, Jacques@ita.br
Instituto Tecnológico de Aeronáutica - ITA – Department of Systems and Control – 12228-900 São José dos Campos - SP

Abstract. *This paper tackles UML modeling and analysis applied to real-time embedded software implementation of a low-cost university satellite 3-axis attitude determination and control system in a hardware-in-the-loop testing environment. The operating system core RTEMS (Real-Time Executive for Multiprocessor Systems) running in a SPARC architecture ERC-32 processor for space applications has been employed for that purpose.*

Keywords*: UML modeling, embedded system design, satellite, attitude determination, control*

1. INTRODUCTION

The main difficulties in developing quality software are requirements specification and conceptual design (Douglass, 2002). Within this context, system modeling has an important role, mainly because it enables project analysis and validation before implementation. In real-time embedded systems, software modeling may be considered as a promising approach to deal with the intricacies that arise in developing such a complex system. Such an approach should provide development flexibility, improved product quality, ease of maintenance, and easier project upgrading, expansion, and reuse of software components (Schulmeyer and McManus, 1992).

Currently considered the main standard for system modeling, the Unified Modeling Language (UML) (OMG, 2005) is a software modeling language that has been widely used in development projects. Its use provides a visual language for modeling, design, and documentation of common artifacts in complex systems software, to better understand systems, and to simplify the reuse of models and source-codes (Heath, 2003). (Note: an artifact is the product of one or more activities within the context of developing a software or system. They are used to capture and convey project information, and can be either a document, or a model, or a model element. Document requirements, or use cases, for example, are artifacts.)

The most common UML-based implementations of embedded system models have been coded with Java or C++, with C language in second place. This is quite surprising because the most common programming language for embedded systems overall by far is C language. Additionally, UML is a software modeling language designed for systems that are based on the paradigm of object-oriented programming, and UML has been used almost exclusively in developing such systems. However, embedded designs are known to be function-oriented (Wang, 2009) and thus a mapping of features from UML onto C language is motivated.

A complete mapping of features from UML onto C language is still emerging. Studies (Wang, 2009, Douglass, 2009) have been performed to facilitate this mapping process, which proposes the use of object-oriented UML notation when the main deployment platform is a function-oriented embedded system to be coded in C language. The establishment of a methodology that tackles the complexities involved in the integration of two different ways of thinking, that is object-oriented and function-oriented, can reduce design time and development, as well as trim down recurrent expenditures due to either bug fixes, or the development of new features.

This paper tackles the modeling of a C-based embedded system using UML for a real-time implementation of a low-cost, university satellite attitude estimation and control system in a hardware-in-the-loop testing environment. Operating system core RTEMS (Real-Time Executive for Multiprocessor Systems) running on a SPARC architecture ERC-32 processor intended for space applications has been employed for that purpose. UML diagrams are used to describe the groundwork for the conceptual construction of the software.

2. EMBEDDED SYSTEM FEATURES

An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions (Kopetz, 1997). By its nature specialist, an embedded system may have numerous applications, for example, automotive systems (engine control, braking systems), computer peripherals (printers, scanners), spaceflight (attitude control system and data management), and myriad others (Heath, 2003).

The main feature of an embedded system, common to all of them, is that these systems manipulate data and interact with the physical real world by controlling some specific hardware. Additionally, (Marwedel, 2003) listed the following features:

- Embedded systems generally interact with the environment, collecting data from sensors and modifying the environment using actuators;

- Embedded systems call for metrics to measure efficiency other than those already known to designers of desktop systems and servers, such as power consumption, code size, performance and cost;
- Some embedded systems have real-time requirements. Not completing a task at a given time may result in data loss, and consequently quality of service in multimedia applications, or cause damage. Failure to comply with a real-time requirement can result in catastrophe; and
- Embedded systems react to the environment, that is, they are in continuous interaction with the environment. One can say that a reactive system is in the state of waiting for an input. For each received input, an embedded system should perform the required processing of information and generate an output.

The presence of features common to many embedded systems justifies the study of development methodologies. Given that the design of a new system can and should use components and modules developed in other projects, software modeling allows development teams to focus on the design features that have not yet been developed, using components from previous projects.

3. SOFTWARE METHODS AND TOOLS

The methods for the development of embedded systems used in this article include a methodology for Hard Real Time System following the paradigm of waterfall lifecycle modeled in UML, with software being coded in C language. The real-time operating system used is RTEMS (Real-Time Executive for Multiprocessor Systems) running in a SPARC architecture ERC-32 processor.

3.1. Real-time operating system

Real-Time Executive for Multiprocessor Systems is a real-time executive (kernel) that provides a high performance environment for embedded military applications including the following features (OAR, 2010):

- Multitasking capabilities;
- Homogeneous and heterogeneous multiprocessor systems;
- Event-driven, priority-based, preemptive scheduling;
- Optional rate monotonic scheduling. A scheduling algorithm used in real-time operating systems with a static-priority scheduling class (Bovet, and Cesati, 2000). The static priorities are assigned on the basis of the cycle duration of the job: the shorter the cycle duration is the higher is the job's priority;
- Intertask communication and synchronization;
- Priority inheritance;
- Responsive interrupt management;
- Dynamic memory allocation; and
- High level of user configurability.

One important design goal of RTEMS is to provide a bridge between two critical layers of typical real-time systems. As shown in Figure 1, RTEMS serves as a buffer between the project dependent application code and the target hardware. Dependency on the hardware specific to a real-time application can be located mostly at the low-level device drivers.

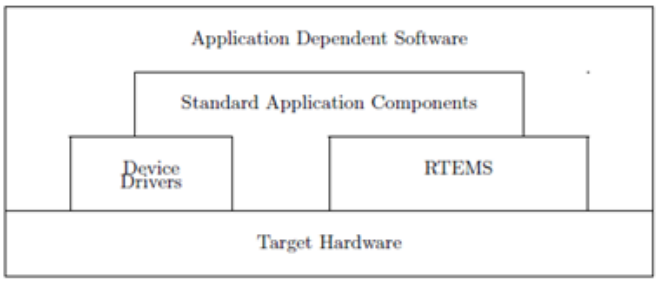

Figure 1. RTEMS application architecture (OAR, 2010 a)

The RTEMS I/O interface manager provides an efficient tool for incorporating such hardware dependency into the system, thus providing a general mechanism to the application code that enables access to the target hardware. A well designed real-time system can benefit from this architecture by building a rich library with standard application components which can be used repeatedly in other real-time projects.

3.2. Lifecycle of embedded software

The development process of embedded software uses the paradigm of waterfall lifecycle. This approach is recommended for systems where security and reliability have great importance. Such trait highlights the quality of a rigid and linear model for developing real-time embedded software in the sense that a phase begins after the previous one has finished (ECSS-E-40, 2008). Inherent to each phase are the procedures of verification and validation, with testing included therein.

To impart the production of embedded software with enhanced flexibility, when necessary, the process lifecycle may make use of a revised waterfall model that leads to the returning to a previous stage in the lifecycle to accommodate functional or technical changes as shown in Figure 2.

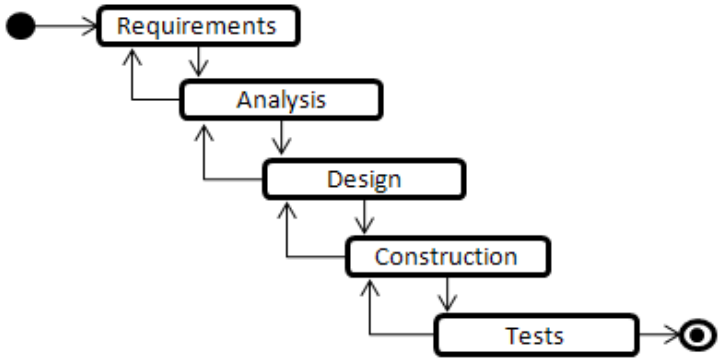

Figure 2. Process lifecycle of embedded software

In the waterfall lifecycle paradigm, requirements specification is the first activity in software development. This stage defines the services that a system should perform, the interfaces with other elements, and the operation constraints (Ferrari and Vincentelli, 1999).

The requirements stage establishes what the system should do rather than how it is to be done. The analysis stage focuses on understanding software behavior in the context of the environment where the system operates, constraints included. The specification capabilities of software modeling languages such as UML are used to allow for the specification of structural and behavioral aspects of the system. The design stage is a multistep process that reinforces four important attributes: data structure, software architecture, procedural detail and interface design. In addition, the design phase aims at translating those requirements into a representation of the software with enough detail for implemention. As in the requirements stage, the design stage also needs documentation, this being done based on the requirements specification.

In the construction phase the programs are coded, the databases created and software modules integrated. The focus of testing is to validate the internal logic of the program procedures, ensuring that all commands have been tested and that system performance produces the expected results for certain inputs.

3.3. Modeling of an embedded system with UML 2.0

UML is a modeling language-oriented paradigm of object-oriented programming (OMG, 2005). Its use provides a visual language for modeling, design, and documentation of common artifacts in complex systems software (Gamma *et al., 1994*), to better understand systems and to simplify the reuse of models and source codes.

While it has not been specifically developed to model real-time systems, some traditional concepts of object orientation, as classes and packages have been improved from the traditional UML. Within this context, three main structures to help real-time modeling can be cited: capsules, ports, and connectors (Douglass, 2002).

- Capsules or active classes represent modules of the architecture, whose only points of interaction with other modules are called ports.
- Ports represent interaction points of a class, or interfaces that specify operations and signals offered by a class, called protocols, allowing communication with the external environment.
- Connectors specify a link that enables communication between two or more entities, the latter being, for instance, either a class, or an object, or an actor in a use case.

UML proposes that modeling be performed through several diagrams that visually show characteristics of the modeled software. As shown in Figure 3, altogether there are eight diagrams divided into two subsets called structural and behavioral diagrams, each with its own function, which together enable the understanding of the system software as a whole.

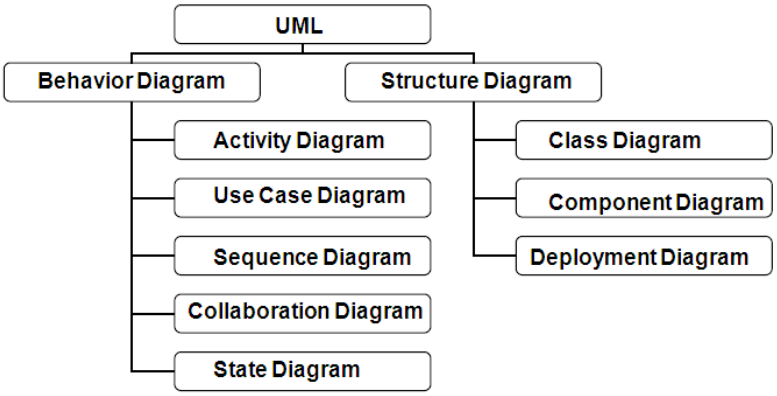

Figure 3. Diagrams proposed by UML (OMG, 2005)

3.4. Mapping from UML onto C

Since modeling language UML has no direct support for C language, this section presents a mapping strategy that defines rules for using UML models with C-language–based embedded systems The primary diagram types defined in this strategy are detailed in Table 1.

Table 1. Mapping from UML onto C: profile diagrams.

UML Basis Diagram	UML C-based Diagram	Description
Use case diagram	Use case diagram	Represents use cases of the system with respect to actors. Creates one .c and one .h file corresponding to each use case.
Component diagram	Build diagram	Shows the set of artifacts constructed from the source file, such as executables and libraries.
Class diagram	Call diagram	Shows calls and their sequences among sets of functions.
Sequence diagram	Message diagram	Represents sequences of calls and events sent among a set of files, including passed parameter values.
State diagram	Statechart	Shows the state machine for a file and how their included functions and actions are executed as events are received.
Activity diagram	Flowchart	Details the control flow for a function or functional unit (such as a use case)

Every use case in the use case diagrams can describe two C files (one .c and another .h header file), each use case defines a behavior yielding an observable result. Sequence diagrams identify the sequence of function calls. In the case where a system object can have different states, as specified in the corresponding State Machine, it can be described using an UML statechart. The Activity Diagram in UML can represent the flowchart for a function use. Event triggers in those diagrams are implemented by triggering the timers, the user inputs, the interrupts, etc (Wang, 2009).

4. FORMULATION OF THE ATTITUDE CONTROL PROBLEM

The satellite attitude dynamics has been modeled as a rigid body with the principal inertia axes as the body reference coordinate frame. It has been assumed that the Attitude Control System (ACS) uses magnetotorquers as the only actuators, and employs Sun sensors and a magnetometer, respectively, to determine the direction of the Sun relative to the satellite, and measure the projection of the Earth's magnetic field along the sensitive axes of the magnetometer. The goal is to keep the solar panels located on the lateral faces of the satellite constantly oriented towards the Sun for adequate power generation and thermal protection of electronic devices at the top and bottom sides of the satellite. The Cartesian coordinate frames that describe the attitude of the satellite and its motion along the orbit are shown in Figure 4.

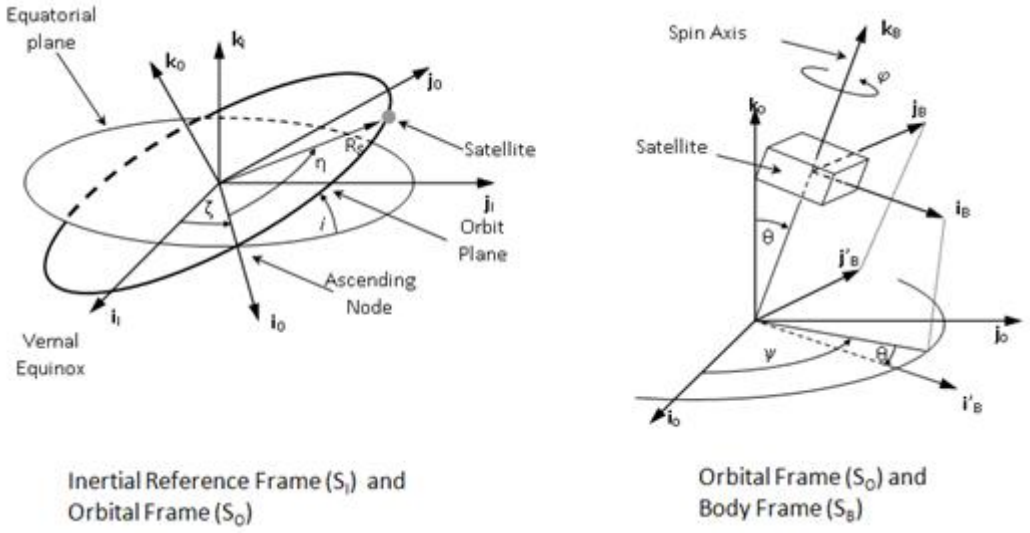

Figure 4. Cartesian coordinate frames (Shigehara, 1972)

R_S is the vector position from the center of the Earth to the satellite center; η is the angle of the satellite position in the orbital plane in relation to the ascending node, ζ is the longitude of the ascending node; and *i* is the inclination of the orbit, with axis $\mathbf{i_I}$ directed to the vernal equinox and axis $\mathbf{k_I}$ perpendicular to the equatorial plane. In the orbital frame, axis $\mathbf{i_O}$ is directed to the ascending node and axis $\mathbf{k_O}$ is perpendicular to the orbital plane. The orbital reference frame and the body coordinate frame are related by the following rotation sequence: rotation ψ around axis $\mathbf{k_O}$, rotation θ around axis $\mathbf{j'_B}$, and spin angle φ around axis $\mathbf{k_B}$. The quaternion has been selected as the attitude parameter in the kinematics differential equation (Chobotov, 1991):

$$\dot{\mathbf{Q}} = \begin{bmatrix} \dot{\lambda} \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 0 & -\omega_i & -\omega_j & -\omega_k \\ \omega_i & 0 & \omega_k & -\omega_j \\ \omega_j & -\omega_k & 0 & \omega_i \\ \omega_k & \omega_j & -\omega_i & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{1}$$

and the real-part component of the rotation quaternion relating inertial and body reference frames is scalar λ; q_1 is the imaginary-part component along the direction of **i**; likewise are q_2 along **j** and q_3 along **k**; and **ω** is the angular velocity of the satellite relative to inertial space.

The implementation of the Attitude Control System (SCA) that is here reported relies on purely magnetic actuation to acquire and maintain the satellite spin axis pointing orthogonal with respect to the ecliptic plane, to mitigate the nutation motion, and to maintain the spin speed in the vicinity of a nominal value, in spite of adversary initial conditions that arise when separation occurs from the launch vehicle upon reaching orbit as a secondary, piggyback payload. Magnetic coils in the magnetotorquers are activated to produce magnetic dipole moments that interact with the geomagnetic field vector during orbital flight and produce small-intensity torques that change the satellite attitude. Attitude acquisition after separation from launch vehicle should not last longer than the available capacity of the batteries on board that supply the SCA operation prior to proper solar panel pointing for battery recharge. Torque coil activation should be timed to consider the magnetometer measurement time window and avoid electromagnetic interference from coil current transients that might compromise the geomagnetic induction measurements, and thus severely degrade attitude and angular rate estimation.

6. MAIN RESULTS

The design of embedded software is based on the classic process of systems development waterfall, with a sequence of well-defined phases, with the possibility of returning to an earlier phase to make changes if necessary. The software architecture has been modeled by integrating concepts used by UML with concepts used by programming language C.

The definition of the software architecture is finalized at this stage of the project. But a flexible software architecture should be able to incorporate changes. So far, the results obtained should be enough to proceed with the design of the software. The architecture is being built covering the strictly necessary to implement an initial version of the software. The evolution of embedded software and changes to novel versions are directly related to verification and validation

through testing to ensure not only the quality of the embedded software, but also that of the architecture. The errors and failures found in the operation of the embedded software indicate conditions that call for improvements that will be implemented in later versions.

6.1. Attitude determination and control system modeling

6.1.1. Use case diagram

The initial step of modeling is the development of the UML use case diagram, which illustrates a high level overview of the software, its functionality, and interaction with the outside world represented by actors. Figure 5 shows this diagram. Table 2 presents a list of actors.

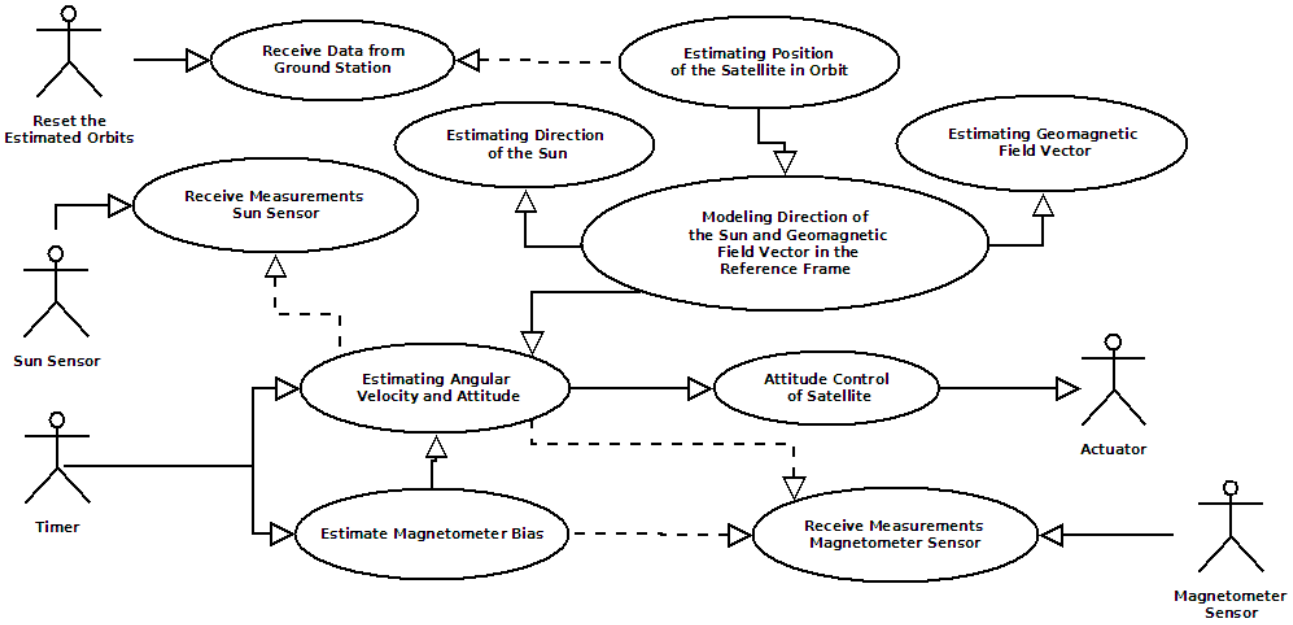

Figure 4. Modeling attitude determination and control system: use case diagram

Table 2. List of actors in the attitude determination and control system model.

Actors	Represented stereotypes
Reset the estimated orbits	Ground station
Sun sensor	Sun sensor, used to determine the direction of the sun relative to the satellite
Timer	Timer task
Actuator	Magnetotorquer
Magnetometer sensor	Magnetometer, used to determine the geomagnetic field vector

The use case initiating the attitude determination and control system model is “Estimating Position of the Satellite in Orbit”, that is to determine the position the satellite in orbit at a precise moment of time. This is defined with parameters of the embedded system, or with updated parameters provided from use case “Receive Data from Ground Station”. Having defined the geo-inertial Cartesian coordinate frame, the geomagnetic field vector and the direction of the Sun are represented in that coordinate frame with the execution of use case “Modeling Direction of the Sun and Geomagnetic Field Vector in the Reference Frame”, composed of the two functions “Estimating Direction of the Sun” and “Estimating Geomagnetic Field Vector”.

Using as reference the modeled geomagnetic field vector and the direction of the Sun, the angular velocity and attitude estimates are computed from use case “Estimating Angular Velocity and Attitude”. Magnetometer bias exerts significant influence on the accuracy of the attitude and angular velocity estimators. Magnetometer bias is therefore estimated with use case “Estimate Magnetometer Bias”.

Magnetometer measurements can be received every second, or every 10 seconds, for magnetometer bias estimation depending on the estimate of angular velocity. Furthermore, magnetometer measurements are received every 100 milliseconds to estimate angular velocity and attitude, along with simultaneous measurements of the Sun sensor, respectively, with use cases “Receive Measurements Magnetometer Sensor” and “Receive Measurements Sun Sensor”.

Control is accomplished by the use case “Attitude Control of Satellite” that computes which coil should be activated and current polarity.

6.1.2. Activity diagram using Flowchart

Once the use cases and actors in the system have been defined, the activity flowchart is used to represent an algorithm, or some detail of the functional flow control, thus indicating in high-level the actions carried out on a given scenario of each use case. Figure 5 shows the activity flowchart that corresponds to use case “Estimating Satellite Position” that estimates the position of the satellite in orbit at a precise moment in time.

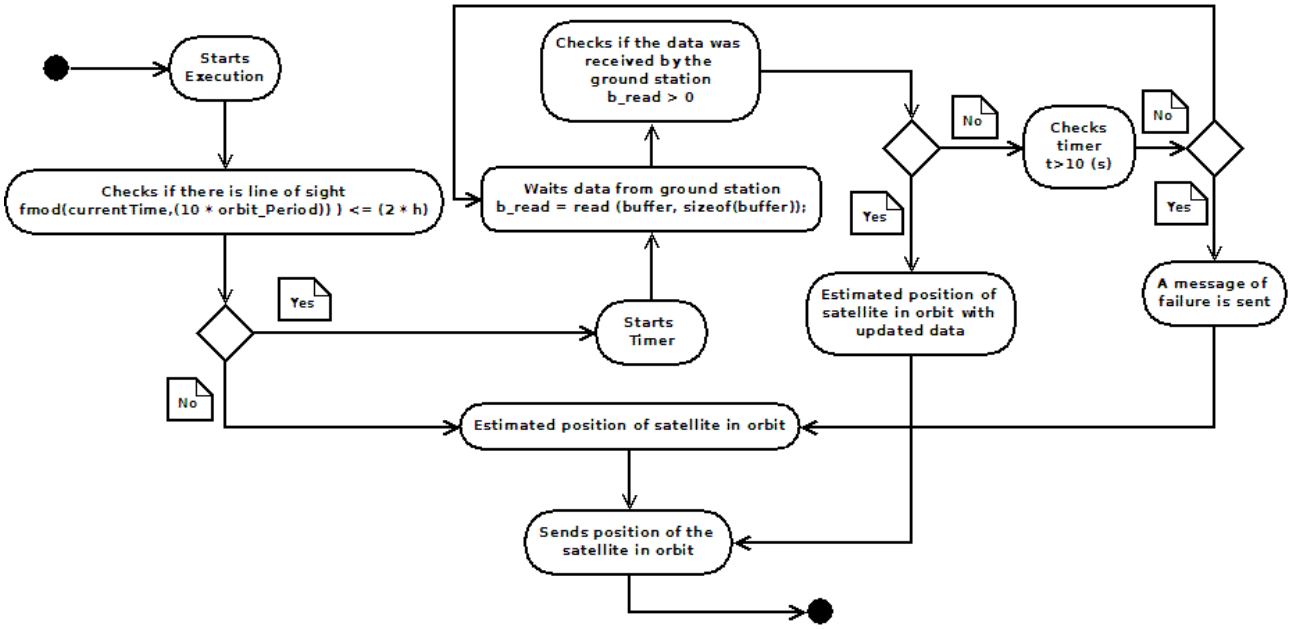

Figure 5. Activity flowchart of “Estimating Satellite Position”

The update of orbit parameters in the embedded system for attitude determination occurs when there is line-of-sight communications from the ground station to the satellite. In case line-of-sight positioning arises from the ground station to the satellite, the embedded system model awaits updated data from the ground for 10 seconds. If errors occur during data transmission and the time limit set for update is exceeded, a message notifying the error is sent, and the required satellite position in orbit is then estimated without use of updated orbit parameters, with the parameters so far stored in the embedded system model. Likewise is the satellite position estimated in case of no line-of-sight with respect to the ground station.

6.1.3. Build diagram

The build diagram represents the artifacts constructed via the compilation and link processes and how they relate to one another. Figure 6 shows the build diagram for the attitude determination and control system. The heard the files, "modulo1.h", "modulo2.h", "modulo3.h", "modulo4.h" and "modulo5.h", contains declaration of variables, functions, and other identifiers. The simulation of sensors and actuators is performed using executables developed in Matlab:

- sensorSolar.mat: simulate sun sensor;
- sensorMag.mat: simulate magnetometer;
- atuador.mat: simulate magnetotorquers; and
- dadoSolo.mat: simulate the ground station.

The circular symbols represent interfaces. For example, "modulo1.h" depends on the interface "Sistema de comunicação" associated with “dadoSolo.mat”.

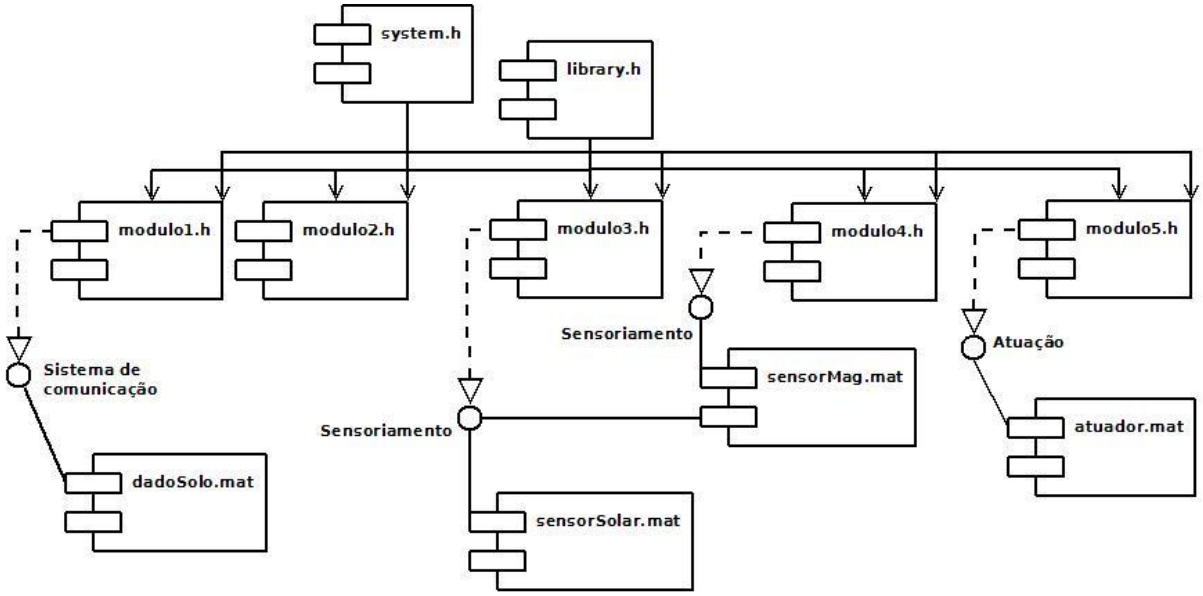

Figure 6. Build diagram

6.1.4. Message sequence

Message diagrams show how the files functionality may interact through messaging by way of synchronous function calls or asynchronous communication. This diagram can be used at different levels of abstraction, and in general several diagrams are needed to represent a use case, because this diagram covers several scenarios of a same use case wherein message exchanges occur. Figure 7 shows one of message diagram for the use case “Estimating Angular Velocity and Attitude”.

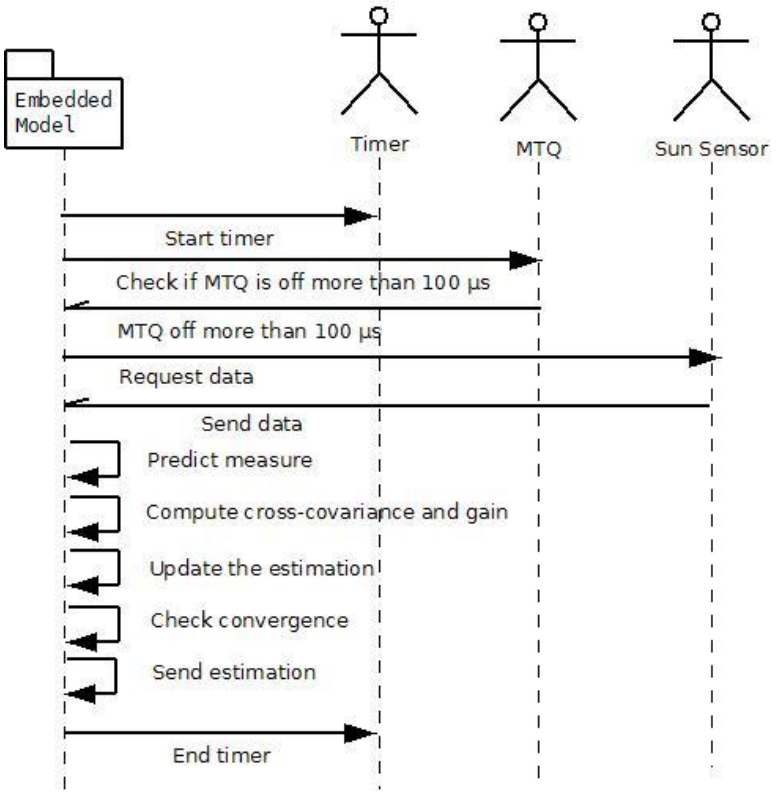

Figure 7. Message diagram

6.1.5. Statechart

A statechart represents the set of states achievable by a file or an use case. There are five files concerning Estimating Position of the Satellite, *init_01.c*, *receberDadosDoSolo.c*, *reset0.c*, *reset1.c* and *task1.c*. Depending on the state of the serial communication, file *receberDadosDoSolo.c* changes its state to one of the following: *Ready*, *Listening*, *Working*, or *Idle*. A statechart of file *receberDadosDoSolo.c* is shown in Figure 8.

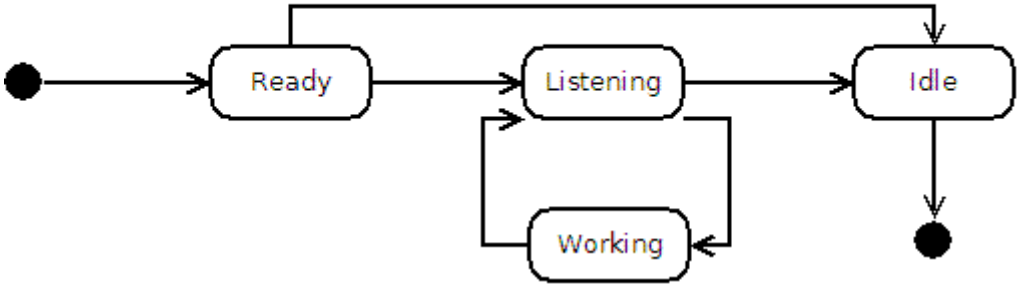

Figure 8. Statechart of file *receberDadosDoSolo.c*

6.2. Software coding and testing

The software-modeling–based implementation is presently under way through C-language coding of the embedded system for satellite attitude estimation and control with real-time operating system RTEMS and its resources. The implementation is being carried out in a SPARC-architecture ERC-32–processor development kit specific for spaceflight applications. The choice of RTEMS as the operating system core arises as a consequence of real-time requirements and costs reduction, since RTEMS is based on the open-source philosophy. The development kit communicates with a host PC that runs a Matlab-based simulation of the differential equations that model orbital flight mechanics, satellite attitude dynamics, and disturbance torques arising from geomagnetic and gravitational perturbations. The implementation is aimed at developing a real-time hardware-in-the-loop simulator on a distributed computing platform that encompasses actual satellite sensors and actuators. The simulator is intended to provide the means for software analysis and testing. Satellite attitude and position in orbit is to be visualized with package Satellite Tool Kit (STK) (AGI, 2005).

7. CONCLUSION

This article tackled modeling and implementation of real-time software for a university satellite attitude determination and control system. The mapping of features from UML onto C reduces design time and development effort, as well as recurrent expenditures due to bug fixes and development of new features when C language is the main deployment platform.

The proposed software model is undergoing implementation using real-time operating system RTEMS. A hardware-in-the-loop simulator is being developed as an alternative testing environment to demonstrate the distributed computing platform developed for the satellite sensor. The definition of the software architecture has been finalized and the results obtained so far are being used to proceed with the real-time embedded system software design.

Finally, the development reported here is part of the ongoing project FINEP/DCTA/INPE Inertial Systems for Aerospace Application with the purpose of providing graduate students at ITA with high-level knowledge and skills in the fields of inertial engineering and real-time systems.

8. REFERENCES

AGI, 2005, “Satellite Tool Kit”, 17 Feb. 2011, < http://www.stk.com.>.

Bovet, D. P., and Cesati, M., 2000, “Understanding the Linux Kernel”, 24 Feb. 2011, http://oreilly.com/catalog/linuxkernel/chapter/ch10.html#85347.

Chobotov, V. A., 1991, “Spacecraft Attitude Dynamics and Control”, Krieger Publishing Company, United States of América, 1ª edition.

Douglass, P.B. , 2002,”Real-time UML developing efficient objects for embedded systems”, Paperback edition, 2ª edition.

Douglass, P.B., 2009, “UML for the C programming language”, Functional-based modeling – IBM.

ECSS-E-40, 2008, "Space Engineering – Software", C Draft 10.0, European Cooperation for Space Standardization (ECSS).
Ferrari, A and Vincentelli, A.F., 1999, "System design: Traditional concepts and new paradigms", Proceedings of IEEE International Conference on Computer Design, pp. 2-13.
Gamma, E., Helm, R, Johnson, R, and Vlissides, J.M., 1994, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley.
Heath, S., 2003, "Embedded System Design", Newnes, San Francisco, 1 edition.
Kopetz, H, 1997, "Real-Time Systems", Kluwer Academic Publishers, Boston MA USA, 1 edition.
Marwedel, P., 2003, "Embedded System Design", Kluwer Academic Publishers, Dortmund, 1 edition.
Schulmeyer, G.G. and McManus, J.I., 1992, "Handbook of Software Quality Assurance", Van Nostrand Reinhold, New York.
Shigehara, M, 1972, "Geomagnetic Attitude Control of an Axisymmetric Spinning Satellite", J. Spacecraft, Vol. 9.
OMG, 2005, "UML - Unified Modeling Language", 17 Feb. 2011, <http://www.omg.org/gettingstarted/what_is_uml.htm>.
OAR, 2010, "RTEMS - Real-Time Executive for Multiprocessor Systems", 10 Feb. 2011, < http://www.rtems.com/>.
OAR, 2010 a, "RTEMS C User's Guide", Edition 4.94, for RTEMS 4.9.4, Vol. 3, pp. 149-167.
Wang, G, 2009, "Modeling C-based Embedded System using UML Design", Proceedings of the IEEE, International Conference on Mechatronics and Automation, Changchun, China.

9. ACKNOWLEDGMENTS

The authors acknowledge the support provided by project FINEP/DCTA/INPE SIA (Sistemas Inerciais para Aplicação Aeroespacial).

10. RESPONSIBILITY NOTICE

The authors are the only responsible for the printed material included in this paper.