European Space Agency Contract Report

The work described in this report was performed under ESA contract. Responsibility for the contents resides in the author or organisation that prepared it.

Low-Cost On-Board Software Development Toolkit

ESTEC/Contract No. 15133/01/NL/ND.

Integration of RTA

Technical Report

WP 3400

Schedulability analysis tools

Version 1.1 — 7 November, 2002

UNIVERSIDAD POLITÉCNICA DE MADRID
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS TELEMÁTICOS

Status: Revised

Author: Miguel Muñoz Arancón

Revised by: Juan Antonio de la Puente

History

Version	Date	Comments
1.0	2002-10-17	First version issued for revision
1.1	2002-11-07	Revised after comments from reviewers

DIT/UPM development team Juan Antonio de la Puente
Juan Zamorano
Alejandro Alonso
Ignacio Martín
Miguel Muñoz
Miguel Ángel Parra

Project Manager Juan Garbajosa
OEI/UPM

Project consortium: **Universidad Politécnica de Madrid**
Departamento de Organización y Estructura de la Información (OEI/UPM)
Departamento de Ingeniería de Sistemas Telemáticos (DIT/UPM).

Techniques Nouvelles d'Informatique

Contents

1 Introduction **1**
- 1.1 Purpose . . . 1
- 1.2 Scope . . . 1
- 1.3 Relation to other projects . . . 1
- 1.4 Glossary . . . 2
  - 1.4.1 Acronyms . . . 2
  - 1.4.2 Definitions . . . 3
- 1.5 References . . . 3
  - 1.5.1 Applicable documents . . . 3
  - 1.5.2 Reference documents . . . 4
- 1.6 Document overview . . . 4

2 System overview **5**
- 2.1 Context, function and purpose . . . 5
- 2.2 The DOBERTSEE environment . . . 5
- 2.3 The RTA tool . . . 6
- 2.4 **Stood** . . . 7
- 2.5 Model description . . . 7

3 Software requirements **9**
- 3.1 Specific requirements . . . 9
  - 3.1.1 Functional Requirements . . . 9
  - 3.1.2 Interface Requirements . . . 10
  - 3.1.3 Operational Requirements . . . 10
  - 3.1.4 Resource Requirements . . . 10
  - 3.1.5 Design and Implementation Constraints . . . 10
- 3.2 Verification, validation and acceptance requirements . . . 11
  - 3.2.1 Verification and Validation requirements . . . 11
  - 3.2.2 Acceptance requirements . . . 11

4 System design overview **13**

5 Architectural design **15**
- 5.1 Architectural desgin overview . . . 15
- 5.2 hood-rta . . . 16
- 5.3 HOOD2TSF . . . 16

5.4 RTA . . . 17
5.5 BrowseCaseml . . . 17

6 Interfaces design **19**
6.1 Interfaces overview . . . 19
6.2 External interfaces . . . 19
6.2.1 Interface to SEE . . . 19
6.2.2 Graphical user interface . . . 20
6.3 Internal interfaces . . . 20
6.3.1 hood-rta . . . 20
6.3.2 HOOD2TSF . . . 20
6.3.3 RTA . . . 21
6.3.4 BrowseCaseml . . . 21

7 Software components detailed design **23**
7.1 Package hood-rta . . . 23
7.2 Package HOOD2TSF . . . 24
7.2.1 Procedure Convert . . . 24
7.2.2 Procedure LoadFile . . . 27
7.2.3 Procedures AnalyzeUpdatedTSF and UpdateTree . . . 27
7.3 RTA binary executable . . . 28
7.4 Reusable package BrowseCaseml . . . 28

A Sample use case **31**
A.1 Introduction . . . 31
A.2 Stood and CASEML HRT-HOOD design files . . . 31
A.3 RTA usage . . . 32

B Software code listing **41**

Bibliography **42**

Chapter 1

Introduction

1.1 Purpose

This document is a technical report about the integration of the RTA schedulability analysis tool in the DOBERTSEE environment, which accomplishes the objectives of WP 3400 in the DOBERTSEE project.

The main objective of the DOBERTSEE project is to produce an affordable, integrated Software Engineering Environment that fully supports the ECSS-E40 processes for developing on-board embedded real-time software.

1.2 Scope

This document applies to DOBERTSEE project, WP 3400. This work package involves the interaction of three tools:

SEE version 1.0. SEE is the integrated development environment which is the product of the DOBERTSEE project.

RTA version 2.1. RTA is a free tool for schedulability analysis of real-time systems which has been developed at DIT/UPM.

Stood version 4.2. Stood is a software design tool for real time and high quality software developments with the HOOD metodology, developed by TNI.

1.3 Relation to other projects

The DOBERTSEE project builds on the results of the following research projects funded by ESA:

- ECSS-PMod, ESTEC contract no. 12798/98/NL/PA, performed by Alenia, GMV and SSF.
- Open Ravenscar Real-Time Kernel, ESTEC contract no.13863/99/NL/MV, performed by DIT/UPM, CASA Space, and the University of York.

The DOBERTSEE RTA tool is based on RTA Version 1.3, which was developed at DIT/UPM as an in-house project. The tool has been adapted so that it can be properly integrated in the DOBERTSEE environment.

1.4 Glossary

1.4.1 Acronyms

CASEML Computer Aided Software Engineering Mark-up Language

DIT/UPM Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid

DOBERTS Dependable on-board embedded real-time systems

DOBERTSEE Dependable On-Board Embedded Real-Time Software

DTD Document Type Definition

ECSS European Cooperation for Space Standardization

ESA European Space Agency

ESTEC European Space Research and Technology Center Engineering Environment

FSF Free Software Foundation

GDB GNU Debugger

GNAT GNU New York University Ada Translator

GNU GNU is Not Unix

GPL GNU Public License

GUI Graphical User Interface

HRT-HOOD Hard Real-Time Hierarchical Object-Oriented Design.

HOOD Hierarchical Object-Oriented Design.

ORK Open Ravenscar Real-Time Kernel

RAVENSCAR Reliable Ada Verifiable Executive Needed for Scheduling Critical Applications in Real-Time

RTA Response time analysis.

SDD Software Design Document

SEE Software Engineering Environment

SRS Software Requirements Specification

TNI Techniques Nouvelles d'Informatique

TNI Techniques Nouvelles d'Informatique.

TSF Task Set File.

URL Uniform Resource Locator

WP Work Package

XML Extensible Mark-up Language.

1.4.2 Definitions

Real-Time System System that is required to react to stimuli from the environment (including the passage of physical time) within timen intervals dictated by the environment. The correctness of a real-time system depends not only on the logical results of the computation, but also on the time at which the results are produced.

Task In Ada, this is the name for a sequential process within a concurrent program.

Lock Any mechanism which provides mutual exclusion for the access to a shared resource in a concurrent program.

Thread In POSIX, independent execution flows within the same program, which have access to communication methods based on shared memory. This opposes to concurrency among programs provided by the operating system.

Response Time Analysis Schedulability analysis method based on the existence of several tasks, which share some resources, and with a defined priority policy.

Rate-Monotonic Analysis Set of techniques based on fixed-priority scheduling theory for the response time analysis of real-time systems.

1.5 References

1.5.1 Applicable documents

P1. Statement of work for low cost on-board software development toolkit. EME/99-133/JAM. ESTEC, 28 February 2000.

P2. Low-cost on-board software development toolkit. OEI-TR-35/2000. UPM/TNI, August 2000

P3. ECSS-E-40B Draft. *Space Engineering. Software.* 28 July 2000.

P4. ECSS-E-40-01 Draft 1. *Space Engineering. Software — Space Segment.* 27 September 2000.

P5. ECSS-Q-80B. *Software Product Assurance.*

P6. ISO 8652:1995. *Ada 95 Reference Manual.*

P7. ISO TR 15942:2000 *Information technology – Programming languages – Guide for the use of the Ada programming language in high integrity systems.*

1.5.2 Reference documents

R1. Tullio Vardanega. *Development of On-Board Embedded Real-Time Systems. An Engineering Approach.* ESA STR-260, 1999.

R2. Alan Burns and Andy Wellings. *HRT-HOOD: A Structured Design Method for Hard Real-Time Ada Systems.* North-Holland, 1995.

R3. Alan Burns. *The Ravenscar Profile*. York University, 2000. Available at `www.cs.york.ac.uk/~burns/ravenscar.ps`

R4. Ada 95 Quality and Style. Springer-Verlag, LNCS 1344. 1995.

R5. STOOD v4.2 User's Documentation. TNI, 2002.

Additional references can be found in the bibliography at the end of this document.

1.6 Document overview

This document is organized as follows:

- Chapter 2 contains a general description of the tools to be integrated at WP 3400, and of the objectives of the integration.
- Chapter 3 lists the software requirements identified as part of WP 3400.
- Chapter 4 an overview of the WP 3400 design.
- Chapter 5 contains the architectural design of WP 3400.
- Chapter 6 specifies the design of the external and internal interfaces of WP 3400.
- Chapter 7 details the design of each of the software components in WP 3400.
- Appendix A contains a case of use of the integrated RTA tool, with detailed operation instructions for the final user.
- Finally, Appendix B gives instructions on where to find the software code listings.

Chapter 2

System overview

2.1 Context, function and purpose

The WP 3400 objective is to integrate a RTA tool in the existing DOBERTSEE environment. This tool allows to analyze the time behaviour of a real-time system using the technique *response time analysis* [12, 13].

The DOBERTSEE environment uses the language CASEML to integrate the data produced by different tools. **Stood** [21] is a tool which implements the HRT-HOOD design method for hard real-time systems. It can generate CASEML files containing all the information of an HRT-HOOD design.

The RTA tool to be integrated is a command-line program which processes task set files. These are ASCII files with a syntax similar to Ada 95, which contain the real-time parameters of a system composed by a set of threads interacting through a series of locks via a preemptive priority based protocol.

The products of WP 3400 are:

- A Tcl package which allows to extract the information needed by the RTA tool from HOOD files written in CASEML format by **Stood** , and vice versa.
- A graphical frontend for the RTA tool, integrated in SEE.
- A reusable Tcl package which simplifies the search for nodes in a CASEML tree.

2.2 The DOBERTSEE environment

The objective of the DOBERTSEE project is to define high-quality Software Engineering Environments (SEE) technology at very low cost, easy to adapt and extend to cover different standards and methodologies, covering the ECSS standard requirements on SEEs and able to run on the Web.

The technical approach is based on a new paradigm for CASE data (CASEML: CASE Mark-up Language), based on Web technology. CASEML has been defined as an extension of XML for the CASE domain. It allows using a single (and extensible) language for the representation of data used and/or generated during the software development process. Being an extension of XML means that any CASE data defined according to a CASEML

DTD can be parsed and validated by a single, simple parser. This in turn implies that the amount of software needed to interpret and manipulate CASEML documents is dramatically reduced, thereby reducing the effort of developing CASEML-based tools and SEEs.

As an integrated environment supporting the ECSS process model, SEE is logically divided into a core that provides basic services on which vertical tools are built, and a set of low-cost tools which provide the basic functionality to the system. One of the basic services implemented by the SEE core is the management of CASEML documents, which allow the data integration among the different tools in the system. WP 3400 bases on this service to communicate the information generated by **Stood** to the RTA tool, and vice versa.

2.3 The RTA tool

RTA is a software tool that performs response time analysis for real-time systems. The analysis is based on fixed-priority scheduling theory [14], and includes most of the techniques which are commonly known as *Rate-Monotonic Analysis* [13].

RTA is portable, and provides a plain text, command-line style user interface. ASCII text files following an Ada 95-like syntax are used as input by the tool, and the tool output is also produced as plain text.

Response time analysis (RTA) is a well known technique for schedulability analysis of preemptive fixed-priority real-time systems [12]. It is closely related to *rate-monotonic analysis* (RMA), which was originally aimed at analysing only cyclic real-time systems, and was later extended to include similar techniques as RTA [13].

RTA is based on a well-defined *computational model*. A computational model defines the execution properties and real-time attributes of all the components of real-time system, as well as the scheduling and synchronization mechanisms that are used at run time [16]. The computational model that will be used for the design of the DOBERTSEE response-time analyser is based on the HRT-HOOD definition [18] and on the report by Vardanega [16]. The main characteristics of this computational model are:

- A real-time system consists of a number of concurrent *threads*. Threads are of two kinds:
  - *Periodic threads* run at regular intervals, triggered by an internal real-time clock.
  - *Sporadic threads* run at unpredictable intervals, triggered by external or internal events. A further distinction between *interrupt-driven* sporadic threads and *software-driven* sporadic threads is often made in order to recognize the different sources of sporadic events which may be present in a real.time system.
- Threads may access shared data with the help of a locking mechanism, such as POSIX mutexes [19] or Ada protected objects [11], which ensures mutual exclusion.

- Threads may synchronize with one another by means of some synchronization mechanism, such as condition variables [19] or entry barriers [11].

  Synchronization is used only as a means for implementing triggering events for periodic or sporadic threads. Blocking synchronization (e.g. as in the classical bounded buffer problem) cannot occur.

  Synchronization objects are dedicated, i.e. no two tasks may wait in the same event.

2.4 Stood

Stood is a software design tool for real time and high quality software developments. It is based on the Hierarchical Object-Oriented Design Method (HOOD) [17], initially created for European Space Agency projects. Also, **Stood** supports HRT-HOOD [18], an extension of the original HOOD method for hard real-time systems. HRT-HOOD adequately addresses the temporal requierments of these systems, by providing analyzable design solutions.

HOOD is an structured software design method, mainly oriented to the Ada programming language. It is based on a simple objects model without inheritance. Objects are organized according to well-known principles:

- Abstraction and information hidding.
- Hierarchy.

Derived from HOOD, HRT-HOOD is oriented to hard real-time systems, which present:

- Strict temporal requirements.
- Critic reliability and security requirements.

It is based on response time analysis methods, and takes into account the temporal and other non-functional requirements of the systems. The execution model used by HRT-HOOD is limited and predictible, in order to build analyzable designs and systems.

In the SEE environment, **Stood** is integrated as a vertical tool used during the architectural and detailed design phases, between requirement analysis and integration. The RTA tool allows to validate the temporal behaviour of the systems designed with **Stood** , as specified by the HRT-HOOD method [18], and completes the temporal parameters of the HRT-HOOD objects with the information extracted from the response time analysis.

2.5 Model description

Figure 2.1 shows the structure and context of the WP 3400 RTA integration in SEE. **Stood** and RTA are two vertical tools integrated in the SEE core and started from the SEE interface, which communicate by means of a CASEML HOOD file.

Therefore, HRT-HOOD designs can be produced using **Stood** and stored as CASEML files. The RTA tool can read those files and analyze its temporal requirements. If the system is schedulable, the RTA tool can complete the temporal parameters of the system by

including the calculated priority or ceiling priority of the terminal objects in the CASEML file. This updated file can be read from **Stood** so that the incremental design process continues.

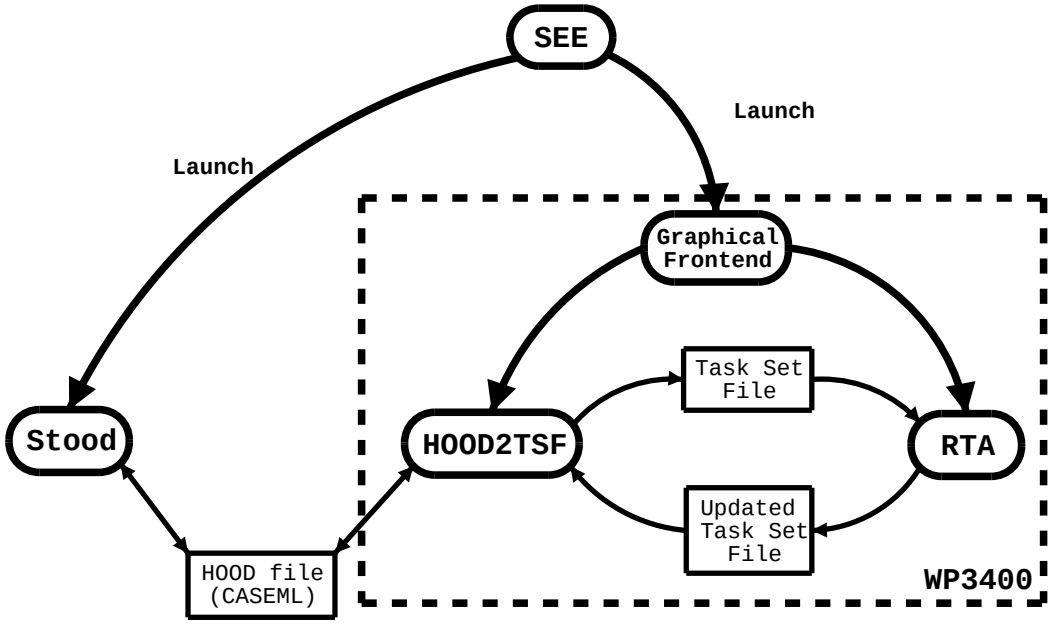

Figure 2.1: WP3400 context and logical model

When the RTA tool is launched, a graphical interface is presented to the user. It allows him or her to open CASEML HOOD files and to start the analysis with RTA. The system design defined in the HOOD file must be expressed as a task set file in order to be analyzed by RTA. The procedures provided by the package HOOD2TSF ar used for that purpose.

RTA is launched then, and its output shown to the user in the graphical interface. Also, RTA updates the contents of the TSF, which is parsed to extract the updated real-time parameters and insert them in the CASEML HOOD file. Again, this process is performed by HOOD2TSF.

Chapter 3

Software requirements

3.1 Specific requirements

This Section details the specific software requirements identified for the development of WP 3400. These requirements refer to the integration of the RTA tool in the SEE, not to the characteristics of the tool itself. See the RTA documentation for detailed information about the RTA tool.

3.1.1 Functional Requirements

RQ-3400-1 A procedure shall be implemented to extract the information relevant to RTA from the HRT-HOOD designs produced by the tool **Stood** . This procedure shall be able to analyze any HRT-HOOD design produced by **Stood** and which contains the necessary information for the temporal analysis.

Inputs. CASEML file containing HRT-HOOD design, created by **Stood** .

Outputs. Real-time parameters of the system.

RQ-3400-2 Task set files compatibles with RTA shall be produced from the real-time parameters extracted from the HRT-HOOD design.

Inputs. Real-time parameters of the system.

Outputs. Task set file.

RQ-3400-3 At the analysis execution, the results shall be retrieved and shown to the user. It shall be clearly specified if the system is schedulable or not.

Inputs. Task set file.

Outputs. Analysis results.

RQ-3400-4 At the analysis execution, the calculated real-time parameters shall be retrieved.

Inputs. Task set file.

Outputs. New real-time parameters (see RTA documentation for details on the calculated parameters).

RQ-3400-5 After the analysis, if the system is schedulable, it shall be possible to update the information listed below in a HRT-HOOD design which will be stored in a CASEML format readable by **Stood** .

- Prioriy of terminal cyclic and sporadic objects.
- Ceiling priority of protected objects.

Inputs. New real-time parameters.

Outputs. Updated CASEML HRT-HOOD design.

3.1.2 Interface Requirements

Three external interfaces can be identified in WP 3400:

- An interface to SEE, which allows to launch the RTA tool.
- External interface of the original RTA tool. This interface is described in the RTA own documentation.
- The user interface.

RQ-3400-6 WP 3400 shall implement a procedure which allows to launch the RTA tool from the SEE environment.

RQ-3400-7 The user interface shall be able to show the detailed RTA results.

RQ-3400-8 The user interface shall allow the user to open CASEML files and save the updated HRT-HOOD designs as CASEML files.

3.1.3 Operational Requirements

WP 3400 supports the iterative design process with HRT-HOOD and **Stood** . The intended use scenario is explained at Section 2.5.

3.1.4 Resource Requirements

The hardware and software resources required by the WP 3400 are the same required by SEE. Refer to SEE documentation for more information.

3.1.5 Design and Implementation Constraints

RQ-3400-9 The WP 3400 will be developed on top of the existing SEE services, which will not be modified.

RQ-3400-10 Tcl/Tk will be used for developing this work package.

3.2 Verification, validation and acceptance requirements

3.2.1 Verification and Validation requirements

The only method identified for the validation and verification of the software requirements is testing. The validation method for each specific requirement are detailed in Table 3.1.

Requirement	Validation Method	Comments
RQ-3400-1	test	
RQ-3400-2	test	
RQ-3400-3	test	
RQ-3400-4	test	
RQ-3400-5	test	
RQ-3400-6	test	
RQ-3400-7	test	
RQ-3400-8	test	
RQ-3400-9	test	
RQ-3400-10	test	

Table 3.1: Specific Requirement to Validation Method Correlation Table

3.2.2 Acceptance requirements

Only the requirements detailed in Table 3.1 will be normative for acceptance.

Chapter 4

System design overview

The product of WP 3400 is an RTA tool integrated as a vertical tool in SEE. The tool selected for the integration is the previously developed RTA, created by DIT as an in-house project. Some Tcl/Tk packages have been developed to allow the communication of RTA and the rest of SEE by means of the CASEML language.

These packages use the services provided by the SEE core for CASEML management. Also, they depend on the reusable Tcl/Tk package SCText, developed as part of WP 3300 (see WP 3300 Software Design Document for details on this package).

A vertical tool in a software engineering environment is an integrated application which supports one particular task in the life cycle of software products. The combination of a core environment and integrated vertical tools has many advantages. On the one hand, the core services provide support of horizontal activities which take place during the whole project, and powerful data integration mechanisms. On the other hand, the integrated tools provide specialization, as well as adaptability and flexibility, because they can be improved, adapted or replaced individually.

In this particular case of WP 3400, the integration of a previously developed and tested tool saves resources in the software development and ensures the quality of the final product.

The internal architecture of the tool, as shown in Figure 2.1, is based on two Tcl/Tk packages, which implement the GUI and the HOOD-TSF interface, and the RTA tool executable. These packages use, in turn, the services implemented by the SEE core, and the reusable Tcl/Tk packages SCText (developed as part of WP 3300) and BrowseCaseml (which is part of WP 3400).

The characteristics of the Tcl/Tk language are ideal for this kind of development. As a scripting language, Tcl provides easy means for the execution of programs. Also, the powerful test processing capabilities allow to manage program input and output, and the graphical capabilities of Tk provide uniform look and feel to the different SEE components. Tcl namespaces are used in WP 3400 to prevent name clashing with the rest of SEE components, as the language lacks of data occultation mechanisms.

Chapter 5

Architectural design

5.1 Architectural desgin overview

The architectural design of WP 3400 is shown in Figure 5.1. It is composed by three Tcl packages and the RTA binary executable.

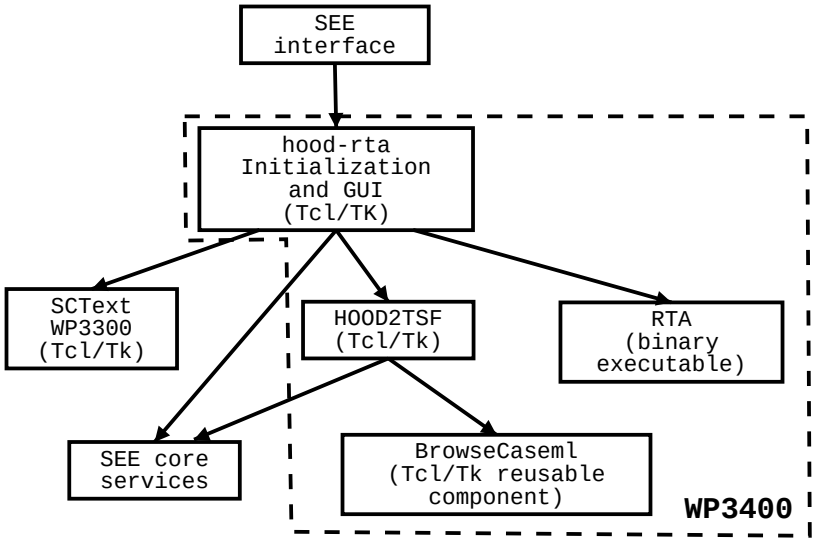

Figure 5.1: WP3400 architecture and context

WP 3400 depends on the SEE core services and the reusable package SCText, which was developed at WP 3300. The Tcl script hood-rta allows to launch the tool from the SEE environment, in a new Tcl interpreter. The packages HOOD2TSF and BrowseCaseml define two namespaces, which export some procedures and variables. The complete architecture is summarized below.

hood-rta (Tcl script) This script allows to lauch the tool in a Tcl interpreter. It creates the GUI, and implements the procedures called from the tool menus.
Subordinates: SCText (WP 3300), File (SEE core), HOOD2TSF, RTA.

HOOD2TSF (Tcl package) This module abstracts the representation of HRT-HOOD designs as needed by RTA (TSF format) and **Stood** (CASEML files), and provides

procedures to translate information between these two representations.
Subordinates: File (SEE core), BrowseCaseml.

RTA (binary executable) It is the executable which actually performs the response time analysis.
Subordinates: none.

BrowseCaseml (Tcl package) This package is intended to be a reusable component which provides some procedures to acceed CASEML trees in a higher level of abstraction than the basic SEE services.
Subordinates: none.

The following sections will detail the characteristics and functionalities of each of these software components, and reference the sowtware requirements that implemented by them.

5.2 hood-rta

hood-rta.tcl is a Tcl script which is used to start the RTA tool, either as a standalone tool or from the SEE environment, by executing:

```
wish hood-rta.tcl
```

It creates a GUI with two elements:

- A Tcl "scrollable text", or non editable text widget with scrollbars (see *WP 3300 Software Design Document*), in which the results of the temporal analyses of HRT-HOOD designs are shown.
- A menu bar, which allows to perform the following operations:
  - Load a HRT-HOOD design from a CASEML file.
  - Start the response time analysis of a HRT-HOOD design.
  - Display help.
  - Exit the RTA tool.

Also, this package defines the Tcl procedures which implement the functionalities available through the menus.

This package implement the functional requirement **RQ-3400-3**, and the interface requirements **RQ-3400-6**, **RQ-3400-7** and **RQ-3400-8**.

5.3 HOOD2TSF

hood2tsf.tcl is a Tcl package which implements the namespace HOOD2TSF. This package provides some procedures that extract the information needed by the RTA tool as TSF files, from the CASEML HRT-HOOD designs produced by **Stood** , and vice versa.

The namespace exports the following elements:

Procedure HOOD2TSF::LoadFile { caseml_file } Reads the CASEML tree from a HRT-HOOD design created by **Stood** .

Procedure HOOD2TSF::Convert Creates a TSF file from the CASEML tree built by the previous procedure.

Procedure HOOD2TSF::AnalyzeUpdatedTSF { fname } Parses a TSF file and extracts the information needed to update the temporal parameters in the CASEML tree containing the HRT-HOOD design.

Procedure HOOD2TSF::UpdateTree Updates the HRT-HOOD design in the tree with the information obtained by the previous procedure.

Variable HOOD2TSF::Error Boolean flag which is set when an error occurs during TSF file creation or response time analysis.

Variable HOOD2TSF::ErrorInfo String which contains the error message generated when an error occurs.

This package implement the functional requirements **RQ-3400-1**, **RQ-3400-2**, **RQ-3400-4** and **RQ-3400-5**.

See Section 6.3.2 for details on the interfaces of the listed procedures.

5.4 RTA

rta is the binary executable which performs the response time analysis of a HRT-HOOD design represented as a task set file.

For the purpose of an automatic analysis of the RTA results, the executable is invoked with the flag "-u", which updates the TSF file with the calculated parameters.

```
rta -u filename.tsf
```

All the information about RTA can be found at [20].

5.5 BrowseCaseml

BrowseCaseml.tcl is a Tcl package that implements the namespace Browse, and exports some procedures to access a CASEML tree in a higher level of abstraction than the basic SEE services:

Procedure Browse::GetChild { Tree node tag } Returns the list of all the childrens of a given "node" which are identified with "tag".

Procedure Browse::GetNode { Tree node taglist } Searches for a list of nodes in "Tree", starting at "node", and following the path specified in "taglist".

"taglist" is a Tcl list containing elements of the form { tag_name } or { tag_name attribute value }.

Procedure Browse::GetValue { Tree node taglist {key ""} {default ""} Searches for a node, as the procedure Browse::GetNode, and returns the value of the attribute "key". If it is not found, the procedure returns "default".

This package is intended to be a reusable component, as it implements some fairly general functionalities.

See Section 6.3.4 for details on the interfaces of the listed procedures.

Chapter 6

Interfaces design

6.1 Interfaces overview

This Chapter identifies and describes the external and internal interfaces of WP 3400.

The external interfaces of this work package are the interface with **Stood** and the rest of SEE, and the graphical user interface. The interaction with **Stood** is based on CASEML HOOD files. The syntax of these files is explained at **Stood** documentation [21], and will not be detailed here.

Regarding to the internal interfaces, the interface provided for each of the components identified in Chapter 5 will be summed up.

6.2 External interfaces

6.2.1 Interface to SEE

The RTA tool, with its graphical frontend, is launched from the SEE environment by loading the script hood-rta.tcl in a new Tcl interpreter. This interpreter must include the same graphical libraries needed by SEE to start [23]. A possible procedure which may be linked to a button or menu in the SEE environment, and used for starting the RTA tool, is shown below:

```
proc Launchers::RTA {} {
    global SEE
    set pwd [pwd]
    cd [file join $SEE(home) tools hood−rta]
    exec wish hood−rta.tcl &                                    5
    cd $pwd
}
```

This procedure changes directory and starts a new Wish interpreter in the background, loading the initialization script hood-rta.tcl, which sets up the GUI of the tool.

The RTA tool requires the services in the SEE library package File [23]. Also, some procedures from the library blt are needed. The restriction to Tcl interpreters which support SEE execution guarantees that this library is available.

The tool may also be started independently from SEE, by loading it in an interpreter compliant to the restrictions mentioned above.

6.2.2 Graphical user interface

The GUI of the RTA tool is structured in the elements listed below:

- A text area in which the results of the temporal analyses of HRT-HOOD designs are shown.
- A menu bar, allowing to perform four operations:
  - Load a HRT-HOOD design from a CASEML file.
  - Start the response time analysis of a HRT-HOOD design.
  - Display help.
  - Exit the RTA tool.

This interface is created by the Tcl script hood-rta.tcl. Also, the procedures implementing the functionalities listed above are declared in this module.

6.3 Internal interfaces

6.3.1 hood-rta

This package does not provide an external interface. It is directly loaded in a Tcl interpreter, as described in Section 6.2.1.

6.3.2 HOOD2TSF

As explained above in this document, the namespace HOOD2TSF exports four procedures and two error variables. Their interfaces are detailed here.

Procedure HOOD2TSF::LoadFile { caseml_file } :
caseml_file: string containing the name of the CASEML file to be loaded.

Procedure HOOD2TSF::Convert :
file: name of the TSF file to be created.

Procedure HOOD2TSF::AnalyzeUpdatedTSF { fname } :
fname: base name of the TSF file to be analyzed.

Procedure HOOD2TSF::UpdateTree :
This procedure has no parameters.

Variable HOOD2TSF::Error :
Value **“1”** means that an error occurred.
Value **“0”** means that there were no errors.

Variable HOOD2TSF::ErrorInfo :
If an error occurred (HOOD2TSF::Error == 1), the corresponding error message is stored in this string variable.

See Section 5.3 for details on the functionality of these procedures.

6.3.3 RTA

The integrated RTA tool provides a command line interface, with a series of flags that allow to specify the behaviour of the tool. For instance, it can be forced to use some user-defined priorities, or calculate them with a deadline-monotonic policy [12]. This interface is explained in detail in the *RTA user's guide* [20].

For the purposes of the integrated tool, the following call to the command-line RTA tool is used:

```
rta -u filename.tsf
```

The flag "-u" stands for update the task set file, that is, the parameters calculated by the tool are updated in the task set file used as input. As this file has a strict syntax, this eases the analysis of the results in order to update the HOOD design in the open CASEML tree.

6.3.4 BrowseCaseml

The BrowseCaseml package is a reusable software component which implements some general operations in a CASEML tree. It provides the interface shown below:

Procedure Browse::GetChild { Tree node tag } :
Tree: CASEML tree to browse.
node: Node in which the search will start.
tag: Identifing tag of the searched node.

Procedure Browse::GetNode { Tree node taglist } :
Tree: CASEML tree to browse.
node: Node in which the search will start.
taglist: List containing the path to the searched node. Its elements are of the form { tag_name } or { tag_name attribute value }.

Procedure Browse::GetValue { Tree node taglist {key ""} {default ""} :
Tree: CASEML tree to browse.
node: Node in which the search will start.
taglist: List containing the path to the searched node. Its elements are of the form { tag_name } or { tag_name attribute value }.
key: Name of the attribute whose value is wanted.
default: Value returned if the node or the tag is not found.

See Section 5.5 for details on the functionality of these procedures.

Chapter 7

Software components detailed design

7.1 Package hood-rta

It is the Tcl package which implements the tool initialization and the creation of the GUI. The tool is started as specified in Sections 5.2 and 6.2.1.

hood-rta.tcl contains a startup script, which loads the needed Tcl libraries and builds the GUI. These libraries are:

- Standard Tcl libraries required by SEE: Tclx, tcllib, BLT.
- SEE library File 1.0, and the rest of libraries needed by it: Tree, CASEMLParse, Entities, Attributes and CasemlUtil.
- Reusable libraries produced at WP 3300 and WP 3400, SCText and BrowseCaseml.
- HOOD2TSF.

The components of the GUI are detailed at Section 6.2.2. Apart from the global variables containing the elements of the GUI, the variable isOpen is declared in this package. It is a boolean flag (with values 1 and 0) that is set when a CASEML HOOD design is opened by the user.

hood-rta.tcl also implements the procedures called from the tool menus. These procedures are:

OpenFile { } Opens a CASEML HOOD design.

CloseFile { } Closes the open CASEML file.

Help { } Displays help to the user.

RTA { } Launches the response time analysis of the open HOOD design.

Exit { } Quits the tool.

7.2 Package HOOD2TSF

The package HOOD2TSF contains the procedures which extract the needed real-time information from CASEML HOOD designs and create TSF files, and which update the information from a TSF file in the CASEML tree.

The namespace defined at HOOD2TSF exports four procedures:

- Convert
- LoadFile
- AnalizeUpdatedTSF
- UpdateTree

7.2.1 Procedure Convert

The algorithm used by the exported procedure Convert to create the TSF file is based on two elements: the object hierarchy tree and the array which contains the information about terminal objects. The steps of this algorithm are listed below:

1. Convert first builds the object hierarchy tree, with the procedure CreateObjectTree.
2. Next, it creates the array of terminal nodes, calling LocateTerminals, and fills the available information in it, with the procedure GetRTInfo.
3. For each cyclic and sporadic object, it calls GetLockInfo to obtain the operations called by the object and their WCET.
4. Finally, it creates the TSF file using the procedure GenerateTSF.

The object hierarchy tree is is a list based data structure. Each of the tree nodes is a list with two elements: { node { children } }. The first element of the pair is the identifier of the OBJECT_DESCRIPTOR tag which contains the specification of a HRT-HOOD object in the CASEML tree. The second element is the list of the node children, expressed in turn as tree nodes. This hierarchy is computed as follows, by the procedure CreateObjectTree:

CreateObjectTree { node }

1. The procedure gets the identifier number of a CASEML OBJECT_DESCRIPTOR tag as a parameter.
2. Child OBJECT_DESCRIPTOR tags of this node are found:
   `$Tree find $node -tag "OBJECT_DESCRIPTOR" -order preorder -depth 1`
   This returns the identifier of the initial node itself, followed by the inmediate children with the tag OBJECT_DESCRIPTOR.
3. Recursively call CreateObjectTree with each found node, and store the results in a list of pairs, called subtree.

4. Return the pair { node subtree }.

Terminal is the array which contains the terminal nodes information. It is a variable accesible from the whole HOOD2TSF namespace. The array has two dimensions, being the first index the identifier number of the CASEML node containing the description of the terminal object. It has the following kinds of elements:

Terminal(id,name) Name of the terminal object.

Terminal(id,type) Type of object. The possible values are "Cy" (cyclic object), "Sp" (sporadic object), "Pr" (protected object), "Pa" (passive object) and "Other" (others, like active objects, which raise an error because a terminal object cannot be active for real time analysis).

For cyclic objects :

Terminal(id,period) Period of the cyclic object.

Terminal(id,deadline) Deadline of the cyclic object.

Terminal(id,offset) Offset of the cyclic object.

Terminal(id,wcet) Worst case execution time of the cyclic object.

For sporadic objects :

Terminal(id,min_arrival_time) Minimun interarrival time between succesive activations of the sporadic object.

Terminal(id,deadline) Deadline of the cyclic object.

Terminal(id,wcet) Worst case execution time of the cyclic object.

Terminal(id,used_ops) For cyclic and sporadic objects, list of the used operations in the format { object_name operation_name operation_WCET }.

The procedure GetLockInfo searches the implementation of the operations called by an object, so that their real temporal parameters can be obtained. It gets the identifier of the caller object, and the calculated objects hierarchy tree, as parameters. Also, it uses the variable Terminal

GetLockInfo { node hierarchy }

1. Find the tag "REQUIRED_INTERFACE" among the node children.
2. Place all the "USED_OBJECT" child tags of the previous one in the list uobjs.
3. For each of the objects in uobjs:
   (a) Place all the "OPERATION" nodes, childs of the "OPERATIONS" tag of the object node, in the list uops.
   (b) For each of the used operations in uops, call the procedure GetOperationData and append the results to the list Terminal(node,used_op).

GetOperationData is the procedure which obtains the real-time parameters from an operation of a given object. Its parameters are the object hierarchy, the initial node containing the object which calls the operation, and the names ob the used object and operation. It uses the algorithm shown next:

GetOperationData { hierarchy ini_node name_obj name_op }

1. Call the procedure FindVisibles to obtain the list of the objects which are visible from the initial one. This is necessary to avoid the mistake with any non-visible operation with the same name as the searched one.
2. Find the object whose name is "name_obj" in the list of visible nodes. Note that the correction of the HRT-HOOD design ensures that this node exists.
3. Using the procedure GetOperationImplementation, find the real-time information of the actual implementation of "name_op", and return its WCET.

In a HRT-HOOD design, an object can call the operations of any node that is sibling of the node itself or of one of its ancestors. The procedure FindVisibles locates the objects accessible from a given one. It has two input parameters (the origin node and the object hierarchy), and two output parameters (a flag indicating if the node was found and the list of its siblings). These two output parameters are passed as variable names, which are imported with the upvar Tcl command.

FindVisibles { node hier found_name visibles_name }

1. Set the output parameter variables to the empty list.
2. If node is the root of the object hierarchy, set the flag found. In this case, the output value of variables is the empty list.
3. Else, for each of the childs in the object hierarchy (let "child_i" be the child node in iteration "i"):
   (a) Recursively call FindVisibles, passing the variable found and a new list variables1.
   (b) If the node was found, return the accessible objects in the variable visibles. These objects are the siblings of child_i and the nodes in visibles1, which are the accessible nodes among the descendants of child_i.

What this procedure does is to traverse the object hierarchy in depth, and once the node is found, go up the hierarchy and accumulate the siblings of the node and each of its ancestors.

Once the called operation is found, finding its temporal parameters requires to locate the operation that actually implements it. In a HRT-HOOD design, the operations of a compound object are implemented by operations of its child objects. The procedure GetOperationImplementation searches the object hierarchy for the operation of a terminal

object which implements a called operation, and returns its WCET. The parameter obj and name_obj are the identifier and the name of the called object, and name_op is the name of the called operation.

GetOperationImplementation { obj name_obj name_op }

1. In the node obj of the CASEML tree containing the HRT-HOOD design, find the tag "INTERNALS > OPERATIONS > OPERATION > IMPLEMENTED_BY" for the operation name_op.
2. If it is found, recursively call GetOperationImplementation with the new object and operation.
3. Else, find the WCET for the given operation at "INTERNALS > OPERATION_ CONTROL_STRUCTURES > OPERATION > REAL_TIME_ ATTRIBUTES > WCET > VALUE".

7.2.2 Procedure LoadFile

This procedure opens a CASEML file using the services provided by the SEE library File. It also checks that the file is a HRT-HOOD design produced by **Stood** , with the heading tag HOOD.

In case an error occurs, this is notified via the variables Error and ErrorInfo, exported by the HOOD2TSF namespace.

7.2.3 Procedures AnalyzeUpdatedTSF and UpdateTree

These procedures are in charge of the updating of the HRT-HOOD design stored as CASEML tree, from the information that the RTA command-line tool writes in the TSF file.

The two procedures share four variables called task_list, lock_list, Task and Lock. The first two are lists which contain the names of the tasks and locks found in the TSF file. The others are bidimensional arrays similar to Terminal. The elements of these arrays are of the form Task(task_name,parameter) or Lock(lock_name,parameter), and they store the following kinds of information:

Task(name,activ): Activation pattern (*periodic* of *sporadic*).

Task(name,prio): Task priority.

Task(name,per): Task period or minimum inter-arrival time.

Task(name,offset): Task offset.

Task(name,jitter): Task jitter.

Task(name,wcet): Task worst case execution time.

Task(name,block): Task blocking time.

Task(name,interf): Task interference time.

Task(name,dline): Task deadline.

Task(name,resp): Task response time.

Lock(name,ceil): Lock ceiling priority.

The procedure AnalyzeUpdatedTSF processes the TSF file updated by RTA with Tcl regular expressions, and fills the contents of these variables.

Next, UpdateTree modifies of creates the tags in the CASEML tree with the priority information: "PRIORITY" for cyclic and sporadic objects, represented as tasks, and "CEILING_PRIORITY" for protected objects, represented as locks.

7.3 RTA binary executable

The internal design of this tool is specified in its own documentation [20].

7.4 Reusable package BrowseCaseml

This Tcl package implements the namespace Browse, which exports three procedures:

- GetChild
- GetNode
- GetValue

GetChild { Tree node tag }

This procedure returns the list of nodes which are children of "node" in the CASEML tree "Tree" which have the tag "tag". It uses the Tcl command:

```
$Tree find $node -tag $tag -order preorder -depth 1
```

GetNode { Tree node taglist }

This recursive function searches for a node in "Tree" starting at "node". "taglist" is a list of elements which can have the form { tag_name } or { tag_name attribute value }. GetNode returns the list of nodes found following the path of nodes indicated by "taglist". This process takes the following steps:

1. Take the first element from tag_list, and assign it to the variable tag.
2. If this element is of the form { tag_name}, call
   GetChild $Tree $node $tag
   If no elements were found, then return the empty list.

3. Else tag is of the form { tag_name attribute value }, then get then call GetChild as in step 2, and filter the nodes whose value for attribute is not the specified one.

4. If tag_list has only one element, return the list calculated in one of the two previous steps.

5. Else, for each element in that list, recursively call GetNode.

6. Concat all the resulting list, remove the duplicates and return the result.

GetValue { Tree node taglist {key “”} {default “”}}

It returns the value contained in the first node found by traversing “Tree”, starting at “node” and following the path “taglist”, which is a list of node tags.

The search successively calls GetChild with the tags in “taglist”, and takes the first element of the result. If the node is not found, the value “default” is returned.

If a value is provided for “key”, the value of the corresponding node tag will be returned. Else, GetValue will return all the information in the node. The Tcl command used for this purpose is:

```
$Tree get $tmp $key
```

That for the default value of “key”, which is an empty string, returns a list of pairs tag-value for all the tags in the tree node.

Appendix A

Sample use case

A.1 Introduction

The RTA tool allows to analyze a HRT-HOOD design to determine if it is schedulable and, in that case, it assigns appropriate priorities to the terminal object in the system. Additional documentation relevant to the RTA user is:

RTA User Guide [20]. This document contains details on the principles on which the RTA is based, and on the strategies used to assign priorities to the system components.

Stood User Guide [21] Instructions to build HRT-HOOD designs analyzable with the RTA tool are found in this document. These design must be exported as CASEML files so that the RTA tool can read them.

A.2 Stood and CASEML HRT-HOOD design files

To export a HRT-HOOD design as a CASEML file from the **Stood** tool, the Sif extractor must be used. The first step is to make an HRT-HOOD design at the graphic editor (see Figure A.1), and complete all the real-time information. This information includes:

- Period and deadline of terminal cyclic and sporadic objects.
- WCET of the threads of cyclic and sporadic objects.
- WCET of the operations called from cyclic and sporadic objects.
- Code of the threads of cyclic and sporadic objects, which include the calls to the operations used by them.

Once the design is completed and saved, at the **Stood** main editor (Figure A.2, select the menu entry "Code->Sif extractor". The dialog box shown at Figure A.3 is displayed, and the root module can be saved.

Once the "OK" button is pressed, the file saving dialog is shown (Figure A.4. The file type CASEML must be selected, and an appropriate filename entered. The resulting CASEML file can be opened from the RTA tool.

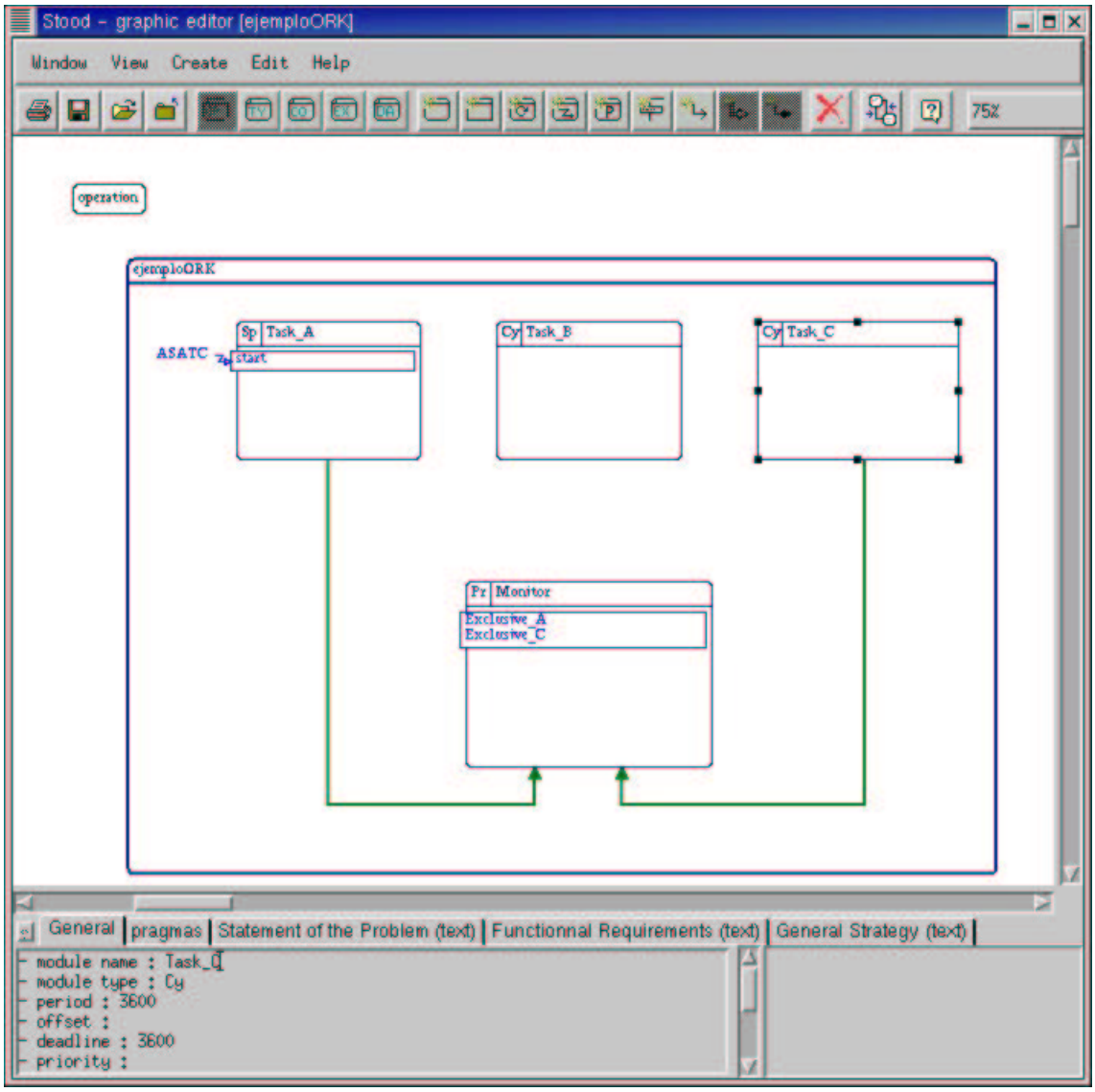

Figure A.1: HRT-HOOD design at **Stood** graphic editor

A.3 RTA usage

The RTA tool can be launched from the SEE environment, or directly from the command line, running:

```
wish hood-rta.tcl
```

When the RTA tool starts, the interface shown at Figure A.5 is displayed.

This interface is composed of a text area and a menu bar. The text area is used to display the results of the schedulability analyses performed by the RTA tool. The scrollbars allow to move the visible text area in order to visualize results which do not fit into the window. The menu bar gives access to the different operations available to the user. Its contents are shown at Figure A.6.

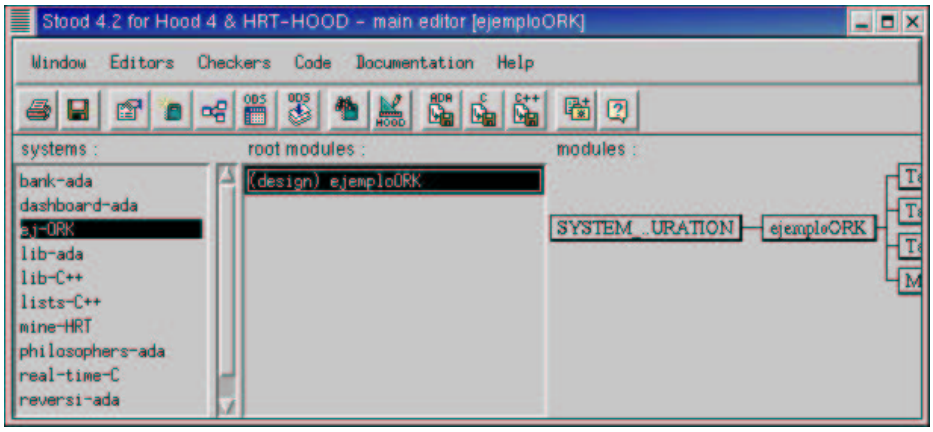

Figure A.2: **Stood** main editor

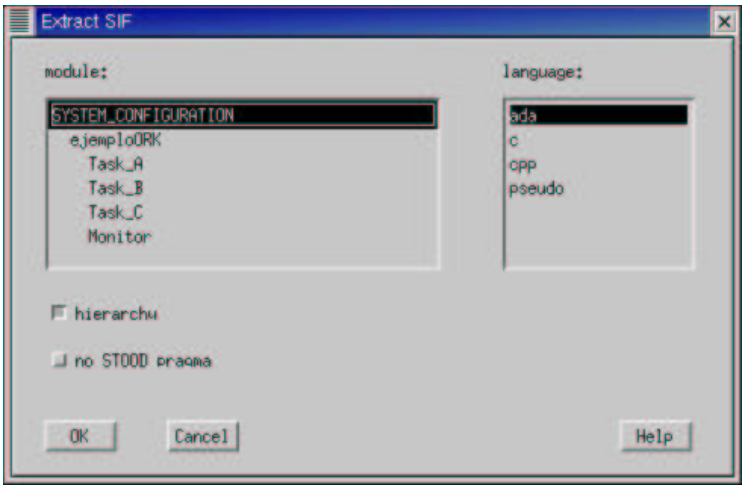

Figure A.3: **Stood** Sif extractor

The File menu is used to open and close CASEML HOOD design files, and to exit the tool. The Tools menu contains the entry Analyze, which allows to launch the schedulability analysis of the open HRT-HOOD design. This entry is disabled if there is no file opened. Finally, the Help menu displays program information to the user. Also, two buttons which give access to the most used program functions are available: the button decorated with a folder is equivalent to the "File->Open..." menu entry, and the one showing a magnifying glass starts the schedulability analysis.

When the "File->Open..." menu is selected, the dialog box shown at Figure A.7 is displayed. A CASEML file containing a HRT-HOOD design.

Once the CASEML file is opened, a confirmation message is displayed in the text area, as shown at Figure A.8. The "Tools->Analysis" menu entry and the corresponding button are enabled, and the user may now launch the response time analysis.

Figure A.9 shows the analysis results for a schedulable HRT-HOOD design. A dialog box informs the user that the design is schedulable, and the RTA output is shown at the text area.

If the analyzed design is schedulable, the user is prompted to update the calculated priorities in the CASEML HOOD file, as shown at Figure A.10.

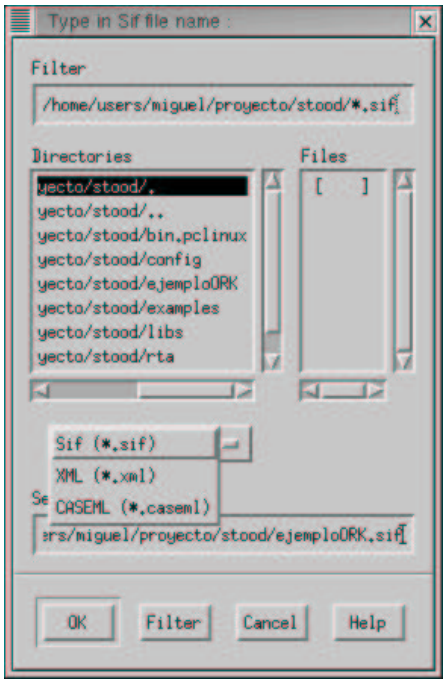

Figure A.4: Export as CASEML

The results shown at the text area (Figure A.11) include the following information:

- **For each task:**
  - Identifier number.
  - Name of the task.
  - Assigned priority.
  - Activation pattern (“S” sporadic, “P” periodic).
  - Task Period or Minimum inter-arrival time.
  - Task Offset.
  - Task Jitter.
  - Task WCET.
  - Blocking time.
  - Task Deadline.
  - Response time.
  - Schedulable task (“Yes”, “No”).

  The tasks are sorted by priority.

- **For each shared resource:**
  - Identifier number.
  - Name of the resource.

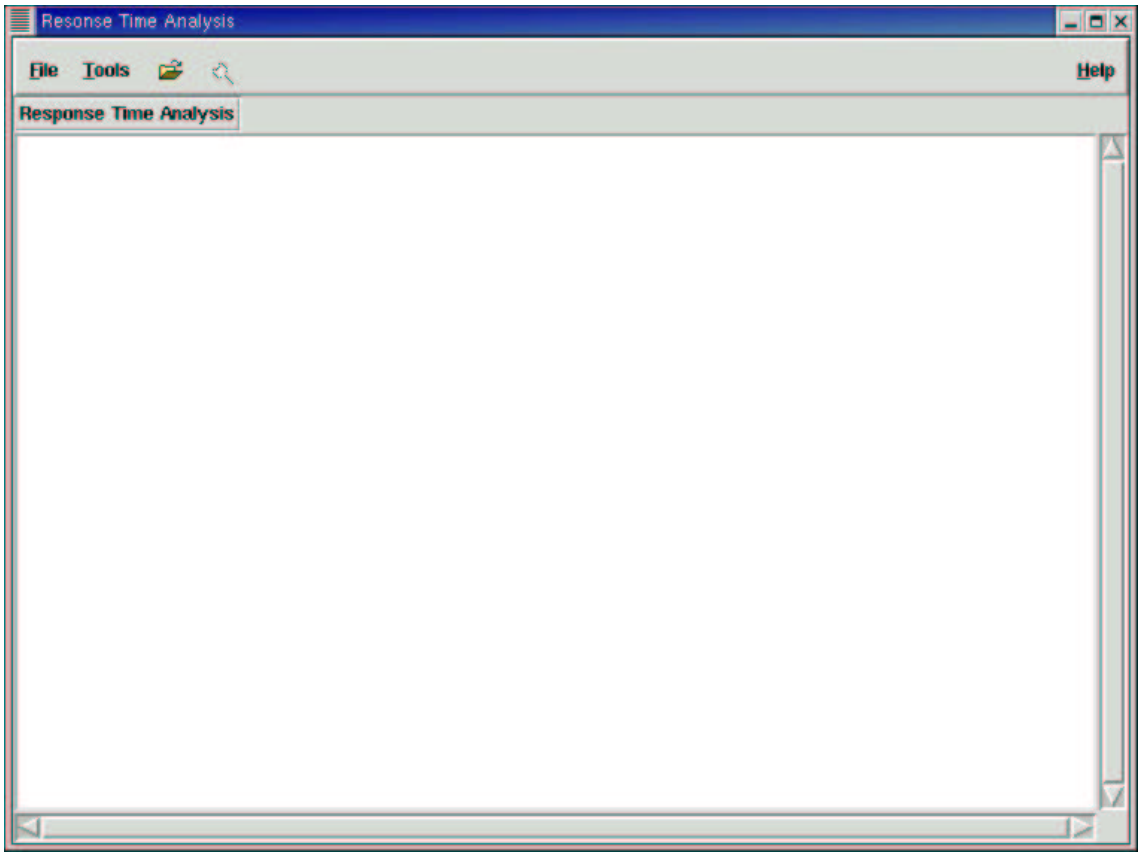

Figure A.5: The RTA user interface

  - Assigned ceiling priority.

  The shared resources are sorted by priority.

- Total processor utilization percentage.

If the design is not schedulable, a message is shown and the text area shows the non-schedulable tasks or the processor utilization highlighted in red colour, as shown at Figure A.12.

An error message as shown at Figure A.13 is displayed if an error occurs during response time analysis. The most common errors are:

- Trying to analyze a CASEML file not containing a HRT-HOOD design. All CASEML files have the extension CASEML, but only HRT-HOOD designs produced by **Stood** can be analyzed.

- Analyzing a HRT-HOOD design whose real-time parameters are incomplete. All the information specified at Section A.2 must be included at the HRT-HOOD design.

- Analyzing a HRT-HOOD design containing active objects. These objects must be decomposed into cyclic, sporadic, protected and passive terminal object in order to perform the response time analysis.

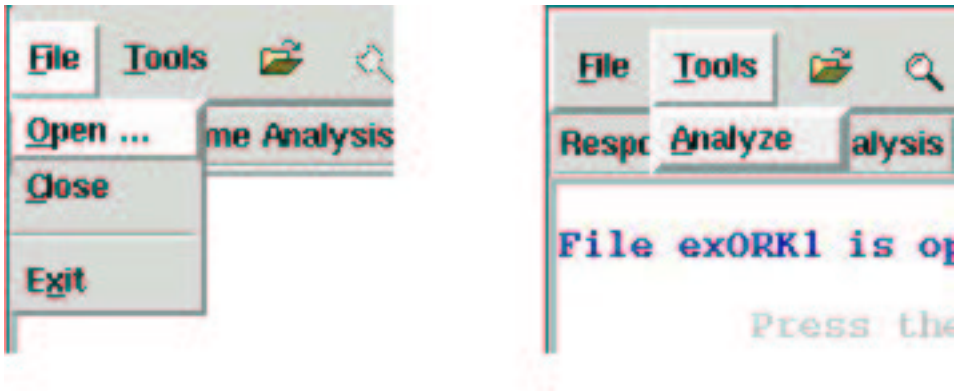

Figure A.6: RTA tool menus

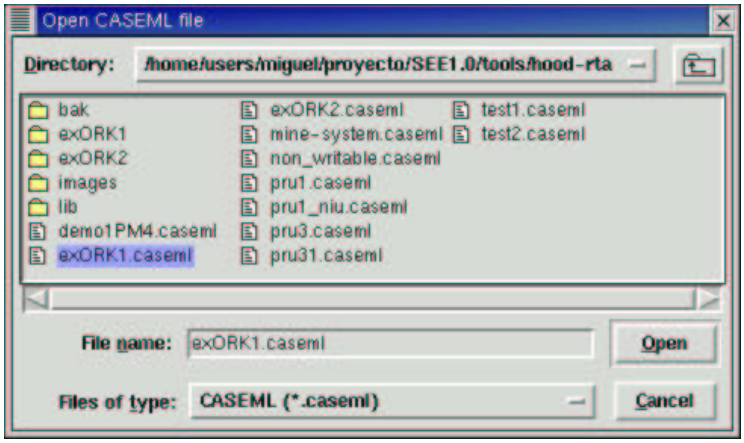

Figure A.7: Open CASEML file dialog box

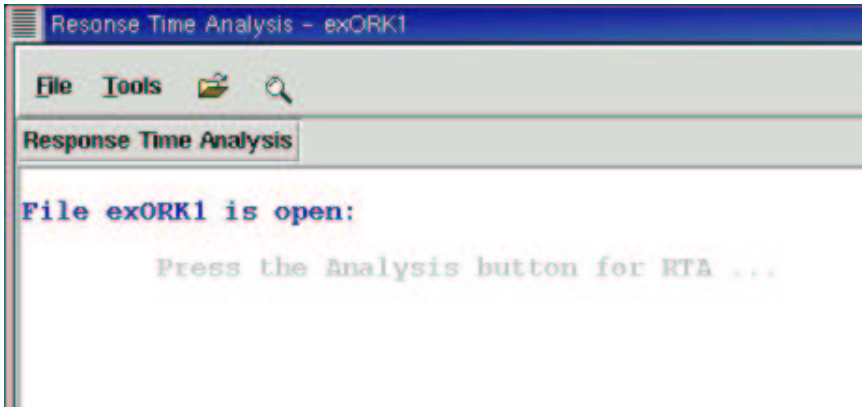

Figure A.8: Opened CASEML file

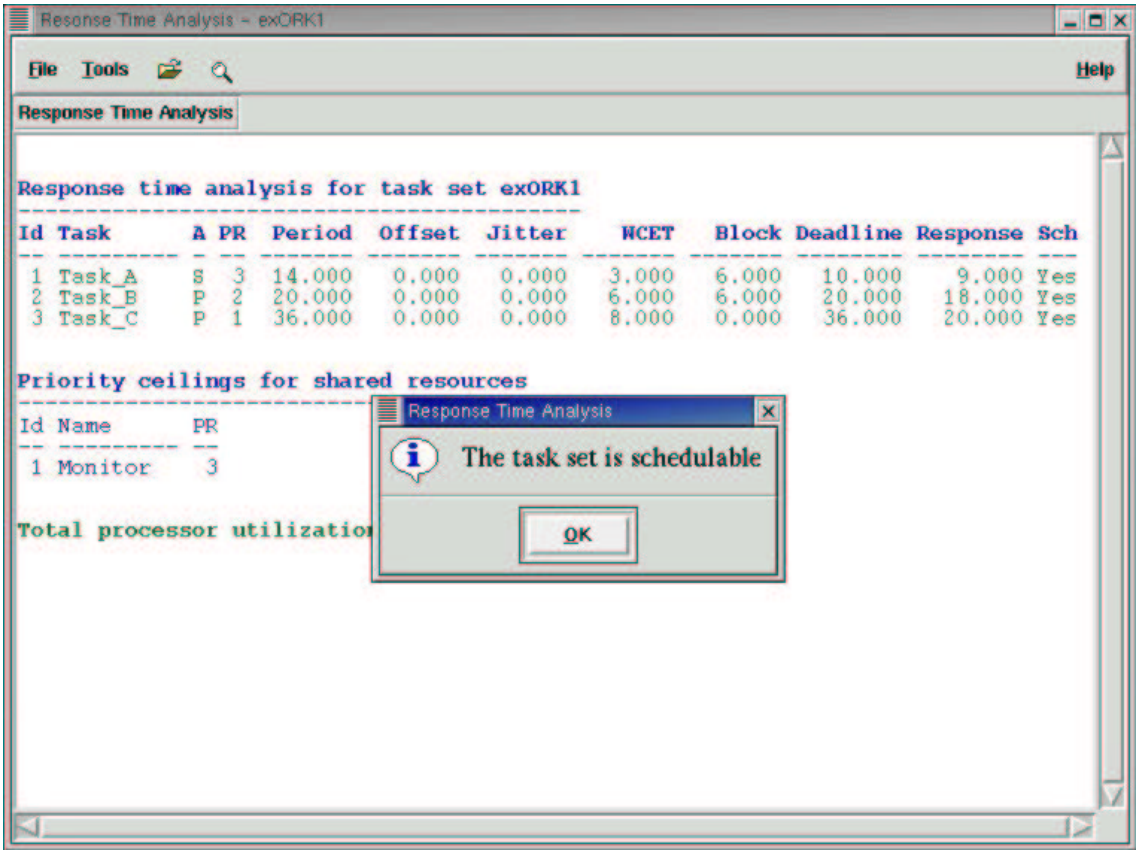

Figure A.9: Results for a schedulable design

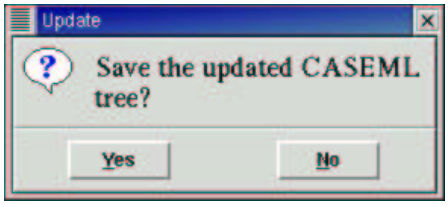

Figure A.10: Update CASEML file dialog box

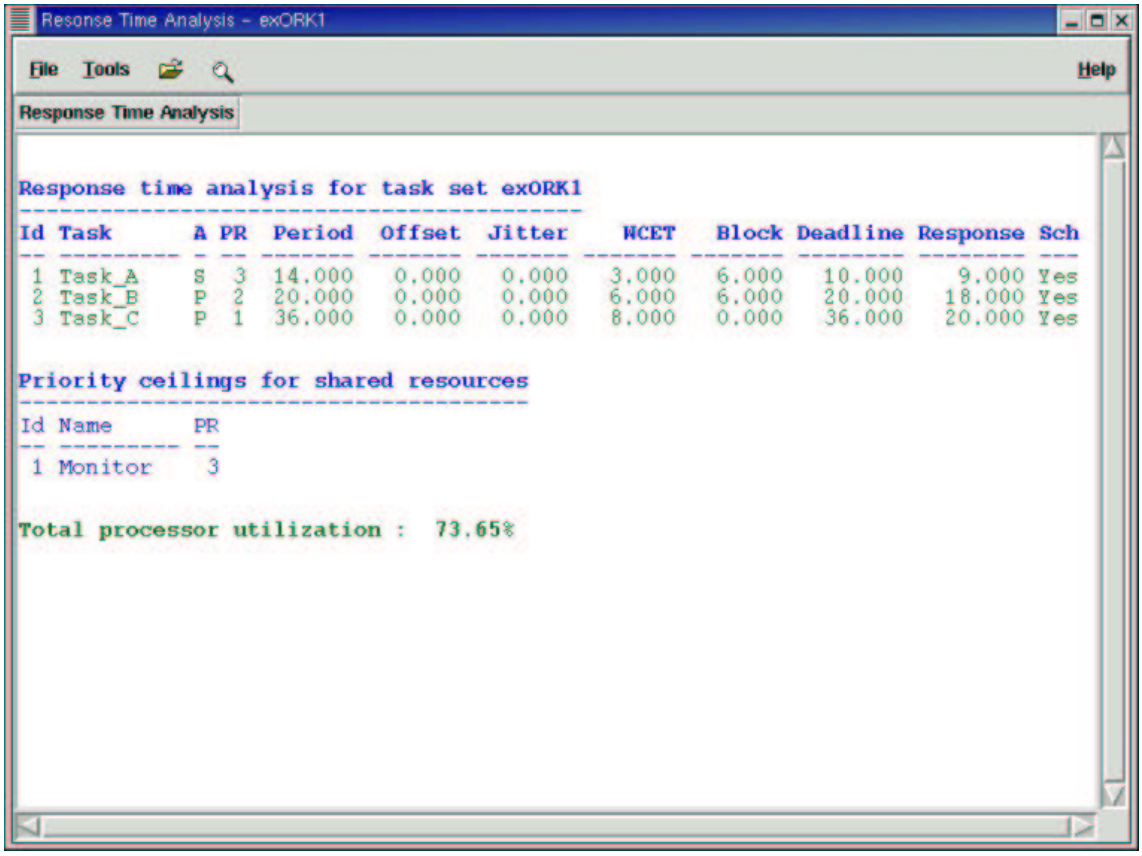

Figure A.11: RTA output

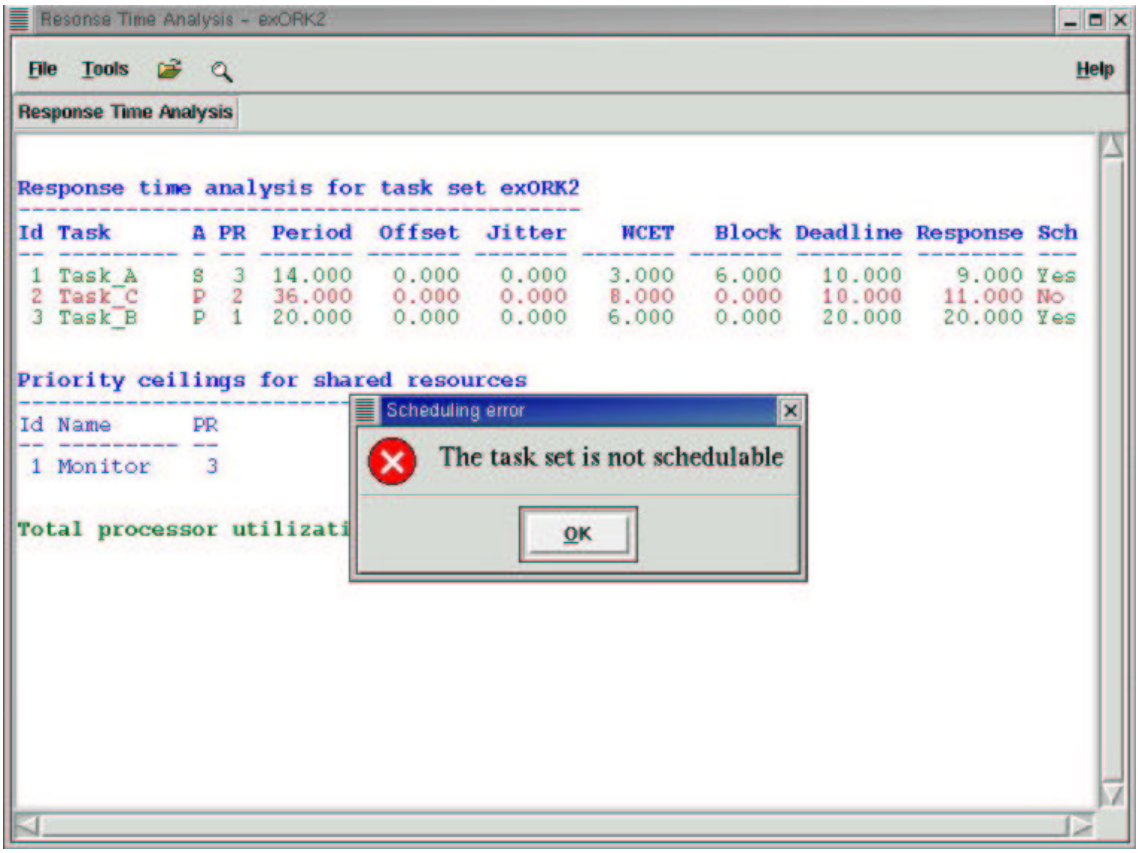

Figure A.12: Results for a non-schedulable design

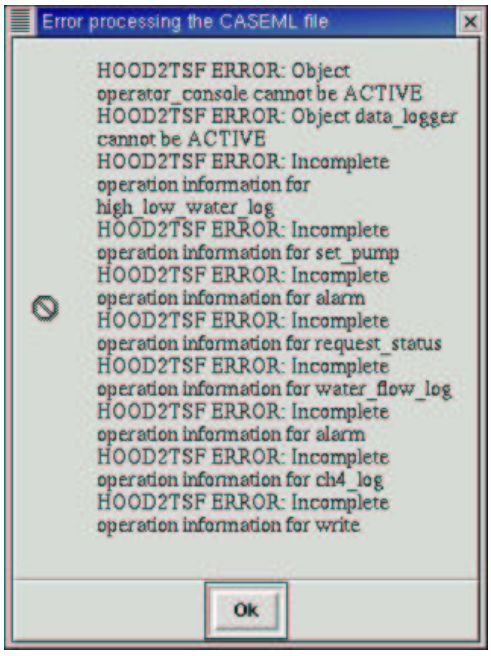

Figure A.13: RTA error message

Appendix B

Software code listing

The documents package WP 3400 are accompanied by a CD-ROM which contains the source code listings.

Bibliography

[1] John K. Ousterhood: *Tcl and the Tk ToolKit.* Ed. Addison-Wesley.

[2] Paul Raines and Jeff Tranter: *Tcl/Tk in a nutshell.* Ed. O'Reilly.

[3] Brent B. Welch: *Practical Programming in Tcl and Tk.* Ed. Prentice Hall.

[4] *Gnat Reference Manual.* Version 3.15a. Ada Core Technologies.

[5] *Gnat User's Guide.* Version 3.15a. Ada Core Technologies.

[6] Open Ravenscar Real-Time Kernel
`www.dit.upm.es/ork`

[7] Proposal made in response to the Invitation to Tender
AO/1-3675/00/NL/FM - Low Cost On-Board Software Development Toolkit from ESA/ESTEC.
`www.dit.upm.es/see`

[8] *ECSS-E40 Standard*. ESA 1999.

[9] *Documentation Templates and Forms for ECSS Software*. ESA 1999.

[10] Juan A. de la Puente, José F. Ruiz, Juan Zamorano Jesús González-Barahona, Ramón Fernández-Marina, Miguel Ánguel Ajo: *Open Ravenscar Real-Time Kernel Operation Manual*. Version 2.2.

[11] *Ada 95 Reference Manual: Language and Standard Libraries*. International Standard ANSI/ISO/IEC-8652:1995, 1995. Available from Springer-Verlag, LNCS no. 1246.

[12] Alan Burns and Andy J. Wellings. *Real-Time Systems and Programming Languages. Addison-Wesley, 2 edition, 1996.*

[13] Mark H. Klein, Thomas Rayla, Bill Pollack, Ray Obenza, and Michael González-Harbour. *A Practitioner's Handbook for Real-Time Analysis. Guide to Rate Monotonic Analysis for Real-Time Systems.* Kluwer Academic Publishers, Boston, 1993.

[14] Neil Audsley, Alan Burns, Rob Davis, Ken Tindell, and Andy J.Wellings. Fixed priority preemptive scheduling: An historical perspective. *Real-Time Systems, 8(3):173-198*, 1995.

[15] John Barnes. *Programming in Ada 95*. 2nd. edition. Addison-Wesley, 1998. ISBN 0-201-34293-6

[16] Tullio Vardanega. *Development of On-Board Embedded Real-Time Systems: An Engineering Approach* 1998.

[17] M. Heitz. *HOOD Reference Manual: Release 4.0*. HRM4-9/22/95, HOOD Technical Group, 1995.

[18] Alan Burns, and Andy J.Wellings. *Hard Real-Time HOOD: A Structured Design Method for Hard Real-Time Ada Systems*. Elsevier, 1995.

[19] *Portable Operating System Interface (POSIX).* IEEE, 1996.

[20] Juan A. de la Puente. *RTA User's Guide*. DIT/UPM, 1998.

[21] *STOOD v4.1 User's Documentation.* TNI, 2000.

[22] L. González, J.Pacios. *SEE User Manual*. GMV, 2000.

[23] *Low-Cost on Board Software Development Toolkit. Design Definition File: Software Design Document. Software Engineering Environment.*