A Case Study for HRT-UML

A Case Study for HRT-UML

Massimo D'Alessandro, Silvia Mazzini, Francesco Donati

Intecs HRT, Via L. Gereschi 32, I-56127 Pisa, Italy
Silvia.Mazzini@pisa.intecs.it

Abstract

The Hard-Real-Time Unified Modelling Language (HRT-UML) method replaces HRT-HOOD with a customized version of UML: it incorporates all the advantages of the HRT-HOOD method and improves the HRT-HOOD design concepts by converging to a more powerful and expressive modelling notation, that is currently a de-facto standard. This paper presents the principles of such a method and the related support toolset and describes the results of a case study showing how the method can be a comprehensive solution to the modelling of Hard Real Time systems.

1. Introduction

The experience shows that the design of Hard Real-Time systems needs methodologies suitable for the modelling and analysis of aspects related to time, schedulability and performance.

In the context of the European Aerospace community a reference methodology for design is the Hierarchical Object Oriented Design (HOOD) and in particular its extension for the modelling of hard real time systems, Hard Real Time-Hierarchical Object Oriented Design (HRT-HOOD), recommended by the European Space Agency (ESA) for the development of on-board systems.

The Unified Modelling Language (UML) is rapidly emerging as the standard object-oriented modelling language. Unfortunately the UML support for developing systems with strict timing requirements is currently poor. This is the motivation for the development of the HRT-UML method.

The HRT-UML method defines an extension profile of Unified Modelling Language, which permits the designer to express the concepts and techniques of the HRT-HOOD method in standard UML. It combines the benefits of a mature, well-understood hard real time design method and an internationally recognized object-oriented design notation.

2. HRT-UML Basics

HRT-UML is a method suitable for the development of Hard-Real-Time systems. It allows the user to:

- design the system taking into account **functional** requirements
- design the system taking into account **timing** requirements
- analyse the **schedulability** of the system
- generate consistent **Ada 95 code**.

2.1 HRT-UML Object Types

HRT-UML objects types represent the different typical HRT behaviour: **Passive**, **Active**, **Protected**, **Cyclic**, and **Sporadic**. They are defined as follows:

- **Passive** - objects that have no control over when invocations of their operations are executed, and do not spontaneously invoke operations in other objects.
- **Active** - objects that may control when invocations of their operations are executed, and may spontaneously invoke operations in other objects. Active objects are the most general class of objects and have no restrictions placed on.
- **Protected** - objects that may control when invocations of their operations are executed, and do not spontaneously invoke operations in other objects; in general protected objects may **not** have arbitrary synchronization constraints and must be analysable for the blocking times they impose on their callers.
- **Cyclic** - objects that represent periodic activities, they may spontaneously invoke operations in other objects, but the only operations they have are requests which demand immediate attention, usually to change the object's execution mode.
- **Sporadic** - objects that represent sporadic activities; sporadic objects may spontaneously invoke operations in other objects; each sporadic has a single operation which is called to invoke the sporadic, and one or more operations which are requests which demand immediate attention, usually to change the object's execution mode.

The hard real-time component of a program will contain at the terminal level only **cyclic**, **sporadic**, **protected** and **passive** objects. Because they cannot be fully analysed, terminal **active** objects are only allowed to design background activities (running at lower priority levels). Active object types, as well as any other object type, may also be used for decomposition of the main system, but they have to be decomposed into cyclic, sporadic or protected, as soon as possible, before reaching the terminal level.

Cyclic and **sporadic** activities are common in real-time systems; each contains a single *thread* that is scheduled at run-time. **Protected** objects control access to data that is shared by more than one thread (i.e. *cyclic* or *sporadic* object); in particular they provide mutual exclusion. **Protected** objects are similar to monitors and conditional critical regions in that they can block a caller if the conditions are not correct for it to continue.

2.2 HRT-UML Real-Time Attributes

HRT projects, objects and operations have associated real-time attributes. Some attributes are concerned with mapping timing requirements onto the design (e.g. the deadline). Other requirements have to be set before the schedulability analysis can be performed (e.g. the worst case execution time). Other attributes are set after this analysis automatically (e.g. the priority).

For each specified mode of operation, the **cyclic** and **sporadic** objects have a number of temporal attributes defined:

- The *period* of execution for each **cyclic** object
- The *minimum inter-arrival time* for each **sporadic** object
- *Offset time* for each **cyclic** object
- *Deadlines* for all **sporadic** and **cyclic** activities.

In order to undertake schedulability analysis, the *worst-case execution time* for each thread and all operations (in all objects) must be known. After the design activity these can be estimated (taking into account the execution environment constraints) and appropriate attributes assigned. The better the estimates the more accurate the schedulability analysis. Good estimates come from component reuse or from arguments of comparison (with existing components on other projects). During detailed design and coding, and through the direct use of measurement during testing, better estimates will become available which typically will require the schedulability analysis to be redone.

During the architecture design activity the designer commits to the run-time scheduling approach, that is the pre-emptive static priority scheduling with priorities assigned using deadline monotonic scheduling theory [Burns 95].

This requires other attributes to be supported: the priority for cyclic and sporadic objects, the *ceiling priority* for protected objects that is assigned at least as high as the maximum priority of the threads that use the operations defined on that object, and the Boolean attribute *schedulable*. Values for these attributes are calculated as a result of the schedulability analysis.

An UML extension profile, that we call HRT-UML, defines all the stereotypes, tagged values and constraints, needed to support HRT-HOOD concepts in UML. The definition is based on the notation defined for stereotype declarations in the OMG-UML [UMLSP 01].

2.3 HRT-UML Design Principles

The starting point for an HRT design is not the definition of classes but rather the set of objects that compose the system and their interconnections. Therefore an HRT-UML system is built directly defining objects and links instead of classes and relationships. This means that the first thing an HRT developer has to deal with is the creation of objects in the system space, without any concern to class issues. Classes of objects become an interesting concern only when some reuse of objects is conceived.

Thus a HRT-UML project is composed of a set of interacting objects which are organized in a structure according to several principles:

- Structural Decomposition - objects may be decomposed into other objects, so that the system can be represented as an *include* graph or *parent-child* hierarchy of objects;
- User-Provider Relations - objects may use services of other objects, so that the system can be represented as a use graph or *user-provider* hierarchy of objects;
- Object Nested Notation - a compact object notation that includes the operation compartment to avoid users to use different diagrams (i.e. class diagrams) and an optional compartment showing the object internals (i.e. the decomposition) to allow users better understand the system topology;
- Multiple Static Instances - objects may have the same structure and properties but differ in their attribute initialisation values and have an independent state evolution. The common structure and properties of such objects is factorised in the *Underlying Class Model*, and an object model, the *Prototype Instance*, from which a new instance may be cloned.

In the HRT-UML approach the class modelling, which is the constructive way to define the system in classical OO methods, is considered as a background model which can be largely derived from the object model in an automatic way.

The HRT-UML toolset allow users to build their system just dealing with objects, automatically creating and maintaining the underlying class model for the purpose of UML meta model integrity. This is done in a transparent way for the users.

When multiple static instances are needed, the class concept comes into picture: **cloning the prototype instance creates a new instance of a class**. The new object may have use relations with required objects in the system, represented as unbound placeholders. They have to be bound (that is redirected) to some real object, which is visible in the instantiation environment.

On the other hand, any object user relation may be made “unbound”, and show its placeholder object, to be bound again to another provider object in the system. The target object of the binding action must obviously be of the same class (interface in the future) of the placeholder object.

3 The Case Study

The case study is based on the re-design of a **Solid State Mass Memory** (SSMM) software. Intecs HRT has originally developed this module for ASI within the frame of the MITA OBDH Enhanced project, to provide storage capabilities for the next small ASI missions.

The SSMM software is in charge of managing the SSMM data storage by writing store pages (SPs) that are based on the type, subtype and application process ID. It provides also support for the transmission to ground of data pages, through a dedicate channel, and functionalities for the SSMM self test.

3.1 High Level Decomposition

The design consists of three active non-terminal objects (ssmmStorage, ssmmCommandManager and ssmmTxManager), a cyclic terminal object (ssmmHk) and two protected terminal (ssmmBoard and ssmmGlobalData).

As introduction to the HRT object notation, note the following:

- suitable stereotypes are used to show HRT nature of objects, they are indicated on the upper left of the object name compartment: A for Active, Pr for Protected, C for Cyclic, etc.
- an operation compartment is added to the objects in order to directly expose the list of operations provided by the object (they correspond to public operations in the underlying class model).
- links indicate use relations: continuous lines indicate direct use relations, while dashed lines indicate intentional use relations, from a non-terminal to a target object, that have to be further satisfied by a direct structural use from a child to the same target.
- in case the UML nested notation is not used for non terminal objects, the {non-terminal} adornment is shown instead in the object name compartment, and their background is grey, instead of white.

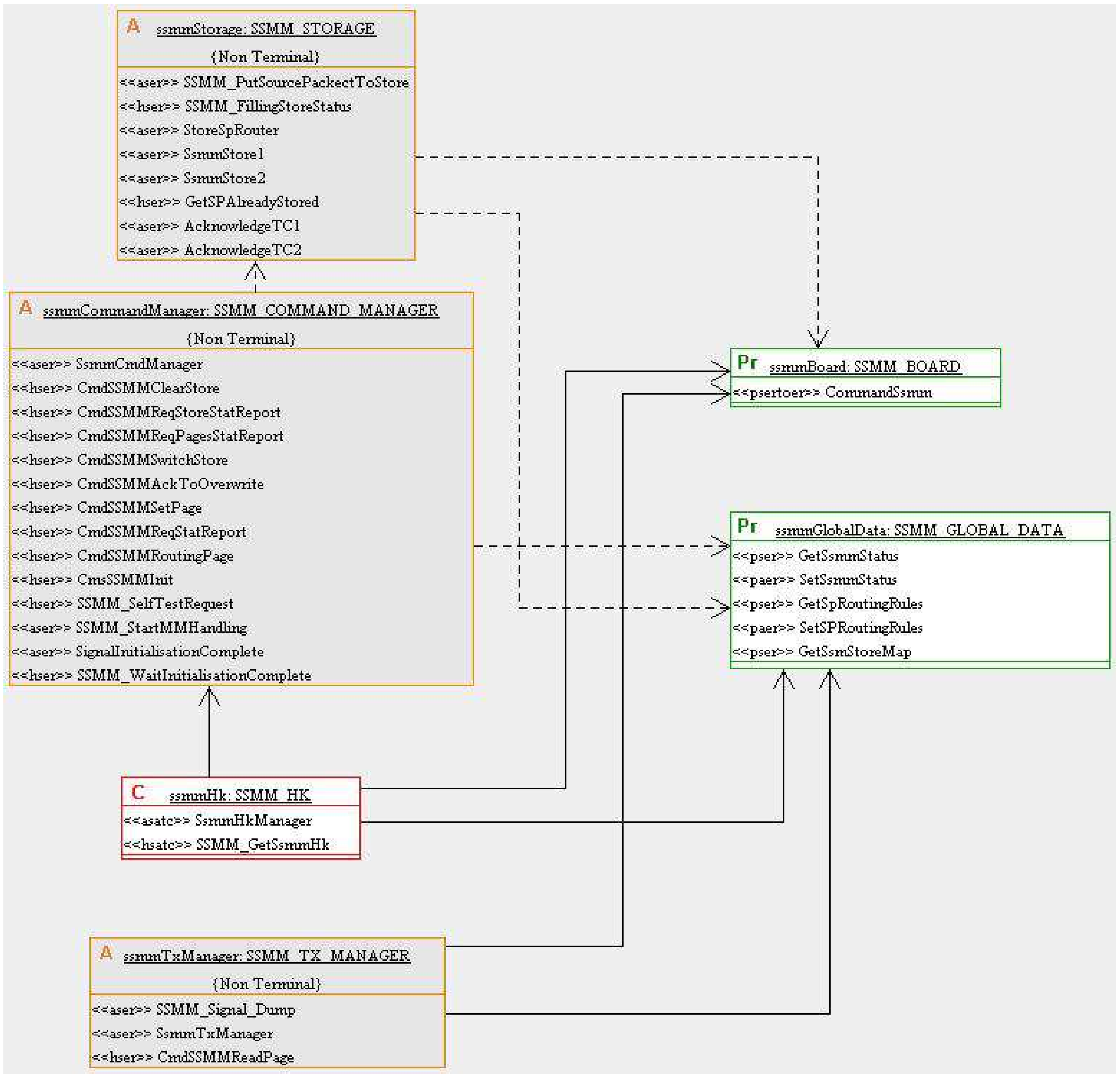
A ssmmStorage: SSMM_STORAGE
{Non Terminal}
<<aser>> SSMM_PutSourcePackectToStore
<<hser>> SSMM_FillingStoreStatus
<<aser>> StoreSpRouter
<<aser>> SsmmStore1
<<aser>> SsmmStore2
<<hser>> GetSPAlreadyStored
<<aser>> AcknowledgeTC1
<<aser>> AcknowledgeTC2
A ssmmCommandManager: SSMM_COMMAND_MANAGER
{Non Terminal}
<<aser>> SsmmCmdManager
<<hser>> CmdSSMMClearStore
<<hser>> CmdSSMMReqStoreStatReport
<<hser>> CmdSSMMReqPagesStatReport
<<hser>> CmdSSMMSwitchStore
<<hser>> CmdSSMMAckToOverwrite
<<hser>> CmdSSMMSetPage
<<hser>> CmdSSMMReqStatReport
<<hser>> CmdSSMMRoutingPage
<<hser>> CmsSSMMInit
<<hser>> SSMM_SelfTestRequest
<<aser>> SSMM_StartMMHandling
<<aser>> SignalInitialisationComplete
<<hser>> SSMM_WaitInitialisationComplete
Pr ssmmBoard: SSMM_BOARD
<<psertoer>> CommandSsmm
Pr ssmmGlobalData: SSMM_GLOBAL_DATA
<<pser>> GetSsmmStatus
<<paer>> SetSsmmStatus
<<pser>> GetSpRoutingRules
<<paer>> SetSPRoutingRules
<<pser>> GetSsmStoreMap
C ssmmHk: SSMM_HK
<<asatc>> SsmmHkManager
<<hsatc>> SSMM_GetSsmmHk
A ssmmTxManager: SSMM_TX_MANAGER
{Non Terminal}
<<aser>> SSMM_Signal_Dump
<<aser>> SsmmTxManager
<<hser>> CmdSSMMReadPage

Figure 1 - Hierarchical First Level Decomposition of the SSMM

3.2 ssmmGlobalData

The ssmmGlobalData object is in charge of managing the global status data for the whole SSMM. It exports some simple functions to read and write data assuring mutual exclusion among the different requests. It is redesigned as an HRT-UML protected resource.

This protected resource contains three different structured data: “ssmmStatusData”, “storeMap”, “spStorageRules”; whose types are defined as global UML Type classes and instantiated inside the Global Data protected resource, as private attributes.

Note that the HRT-UML does not provide public attributes neither in the notation neither in the model. This leads to a more compact and sounder design, where all the public services are operations while data is encapsulated within HRT objects (usually protected objects).

3.3 ssmmTxManager

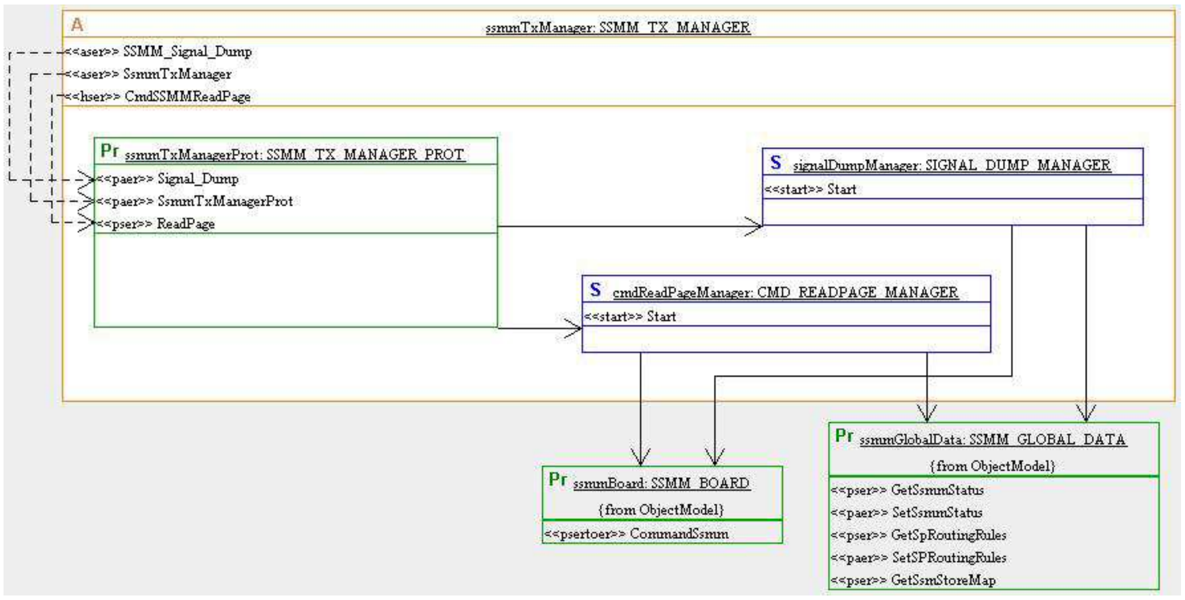
A
ssmmTxManager: SSMM_TX_MANAGER
<<aser>> SSMM_Signal_Dump
<<aser>> SsmmTxManager
<<hser>> CmdSSMMReadPage
Pr ssmmTxManagerProt: SSMM_TX_MANAGER_PROT
<<paer>> Signal_Dump
<<paer>> SsmmTxManagerProt
<<pser>> ReadPage
S signalDumpManager: SIGNAL_DUMP_MANAGER
<<start>> Start
S cmdReadPageManager: CMD_READPAGE_MANAGER
<<start>> Start
Pr ssmmGlobalData: SSMM_GLOBAL_DATA
{from ObjectModel}
<<pser>> GetSsmmStatus
<<paer>> SetSsmmStatus
<<pser>> GetSpRoutingRules
<<paer>> SetSPRoutingRules
<<pser>> GetSsmStoreMap
Pr ssmmBoard: SSMM_BOARD
{from ObjectModel}
<<psertoer>> CommandSsmm

Figure 2 - ssmmTxManager

The ssmmTxManager component manages the transmission requests from ground as sporadic events. The transmission related TCs may either enable the dumping of one (or more) whole store(s) or request the transmission of a set of pages.

The component is decomposed in three objects: a protected resource ("ssmmTxManagerProt") and two sporadic tasks ("signalDumpManager" and "cmdReadPageManager"). The protected one receives requests through his interface's methods ("Signal_Dump", "ReadPage"). Each method calls the associated sporadic task, "signalDumpManager" for the method "Signal_Dump" and "cmdReadPageManager" for the "ReadPage" method.

3.4 ssmmStorage

The ssmmStorage component is in charge of performing the SPs storing in SSMM. Different source packets in type, subtype and application process identifier are written in different stores with a prioritised order on the basis of SPs routing rules.

In the HRT-UML solution active tasks are designed as sporadic, while queues are still protected resources.

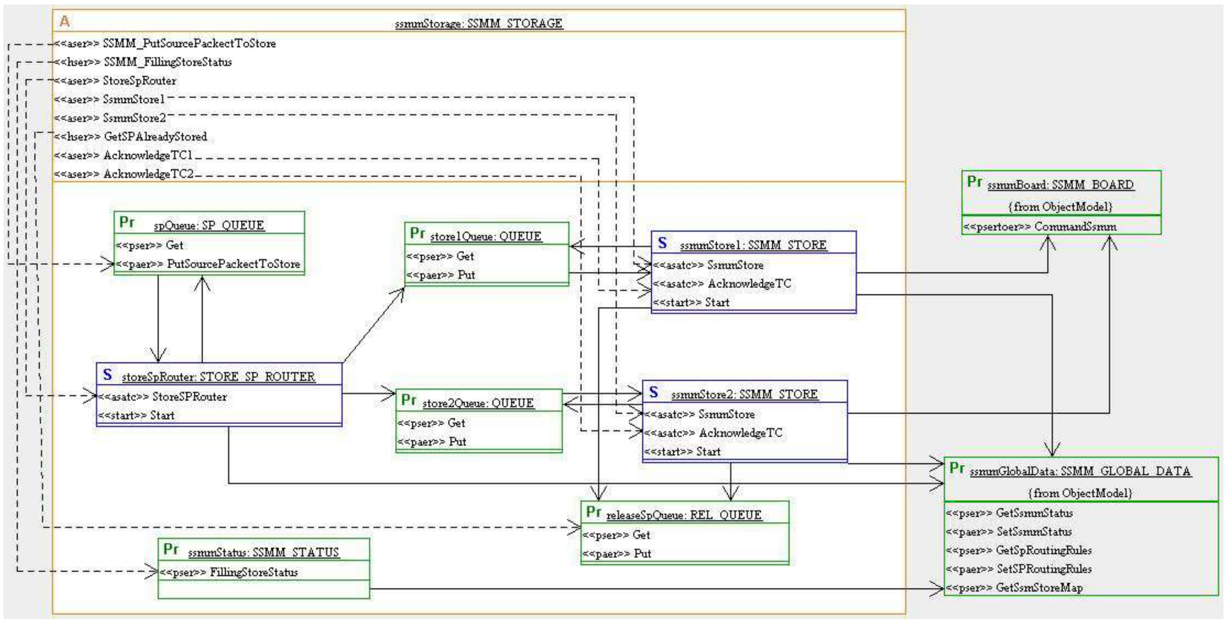

Figure 3 - ssmmStorage

Actions sequence:

- When calling the SSMM_PutSourcePacketToStore method, stores are put in the entry queue (ssmmSpQueue) and the sporadic task StoreSpRouter is also activated.
- the sporadic task StoreSpRouter when activated does the following:
  - takes a store from the queue through a call with timeout
  - chooses the destination queue (store1Queue or store2Queue), moves the store and activates the sporadic task associated to the queue.

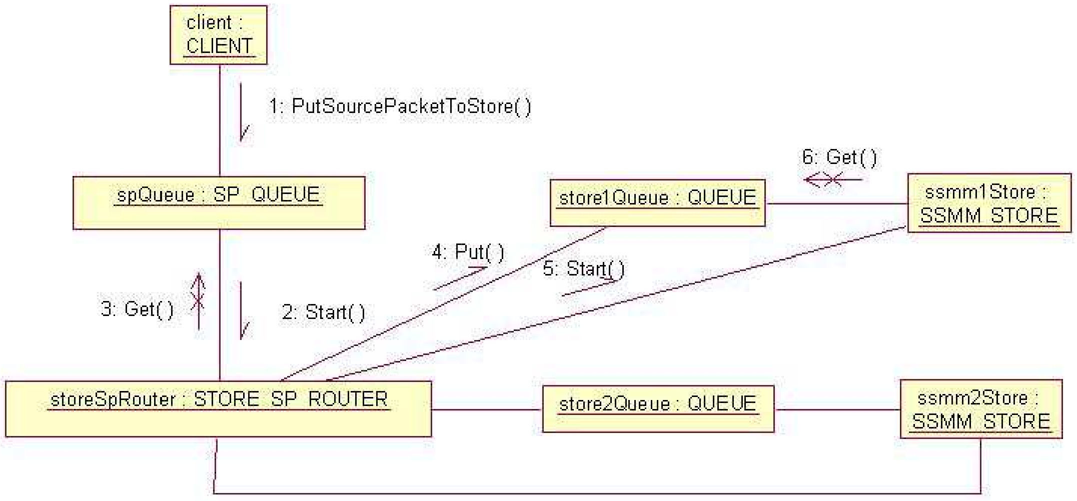

Figure 4 - SSMM_Storage collaboration scenario

ssmmStore1 and ssmmStore2 work in the same way moving the stores from their entry queue, store1Queue and store2Queue respectively, to the SSMM_BOARD.

Note how the HRT dimension has deep impact on the architectural choice and how HRT-UML suitably support both the reasoning about the problem and the expression of the chosen solution.

3.5 ssmmHk

This module is designed as a cyclic task. It is in charge of acquiring the SSMM board housekeeping data at a fixed frequency (1 Hz) and of providing the overall housekeeping data to the OBDH).

It allows the execution of housekeeping data collection also during self test or init. It has the maximum priority among the ones reserved for SSMM SW.

3.6 ssmmCommandManager

This module is the main interface for OBDH commands. It provides functionalities for SSMM initialisation, stores clearing, and self-test. It also manages some telecommands to modify the status of stores and pages. It is decomposed in a protected task, which receives command requests from the external environment ("ssmmCommand ManagerProt"), and four sporadic tasks that are in charge of executing complex commands.

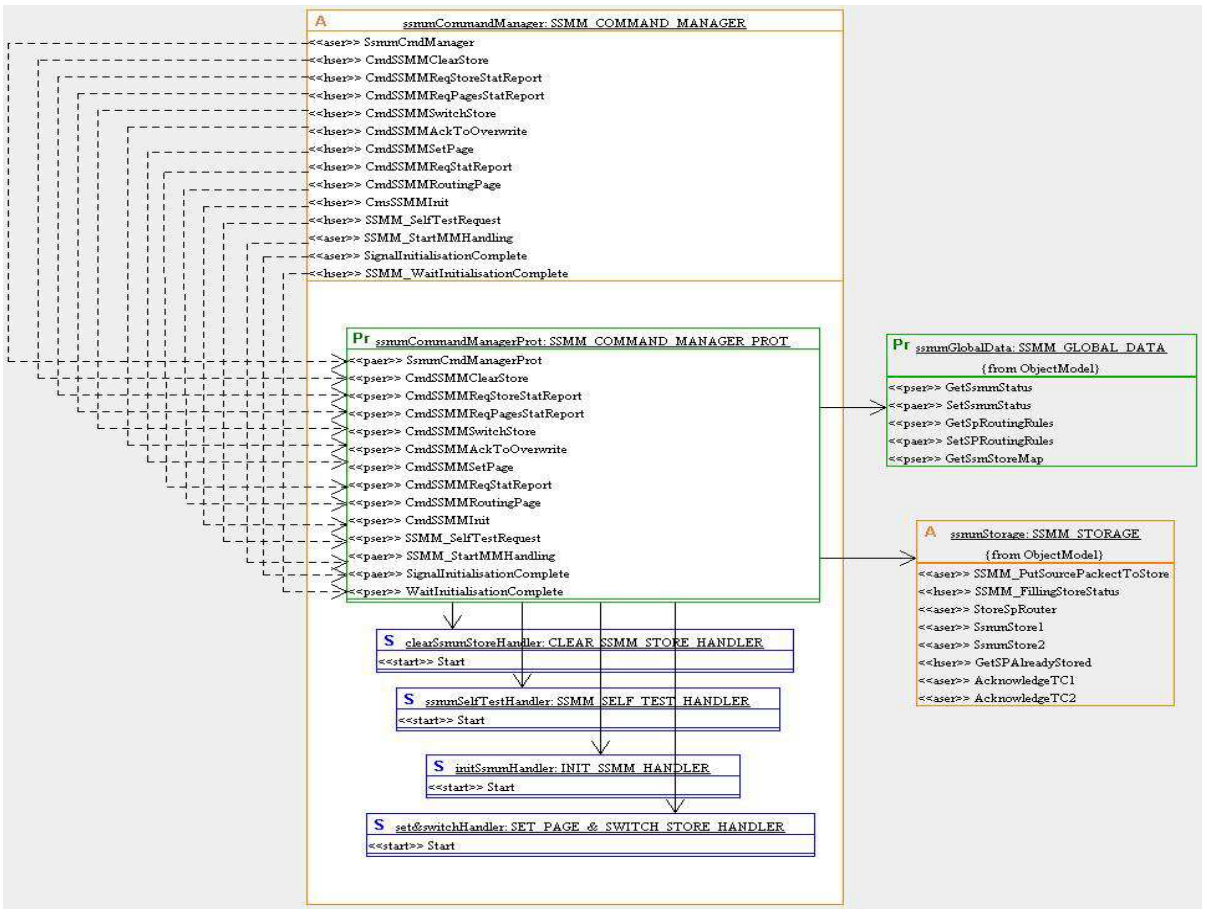
A ssmmCommandManager: SSMM_COMMAND_MANAGER
<<aser>> SsmmCmdManager
<<hser>> CmdSSMMClearStore
<<hser>> CmdSSMMReqStoreStatReport
<<hser>> CmdSSMMReqPagesStatReport
<<hser>> CmdSSMMSwitchStore
<<hser>> CmdSSMMAckToOverwrite
<<hser>> CmdSSMMSetPage
<<hser>> CmdSSMMReqStatReport
<<hser>> CmdSSMMRoutingPage
<<hser>> CmsSSMMInit
<<hser>> SSMM_SelfTestRequest
<<aser>> SSMM_StartMMHandling
<<aser>> SignalInitialisationComplete
<<hser>> SSMM_WaitInitialisationComplete
Pr ssmmCommandManagerProt: SSMM_COMMAND_MANAGER_PROT
<<paer>> SsmmCmdManagerProt
<<pser>> CmdSSMMClearStore
<<pser>> CmdSSMMReqStoreStatReport
<<pser>> CmdSSMMReqPagesStatReport
<<pser>> CmdSSMMSwitchStore
<<pser>> CmdSSMMAckToOverwrite
<<pser>> CmdSSMMSetPage
<<pser>> CmdSSMMReqStatReport
<<pser>> CmdSSMMRoutingPage
<<pser>> CmdSSMMInit
<<pser>> SSMM_SelfTestRequest
<<paer>> SSMM_StartMMHandling
<<paer>> SignalInitialisationComplete
<<pser>> WaitInitialisationComplete
Pr ssmmGlobalData: SSMM_GLOBAL_DATA
{from ObjectModel}
<<pser>> GetSsmmStatus
<<paer>> SetSsmmStatus
<<pser>> GetSpRoutingRules
<<paer>> SetSPRoutingRules
<<pser>> GetSsmStoreMap
A ssmmStorage: SSMM_STORAGE
{from ObjectModel}
<<aser>> SSMM_PutSourcePacketToStore
<<hser>> SSMM_FillingStoreStatus
<<aser>> StoreSpRouter
<<aser>> SsmmStore1
<<aser>> SsmmStore2
<<hser>> GetSPAlreadyStored
<<aser>> AcknowledgeTC1
<<aser>> AcknowledgeTC2
S clearSsmmStoreHandler: CLEAR_SSMM_STORE_HANDLER
<<start>> Start
S ssmmSelfTestHandler: SSMM_SELF_TEST_HANDLER
<<start>> Start
S initSsmmHandler: INIT_SSMM_HANDLER
<<start>> Start
S set&switchHandler: SET_PAGE_&_SWITCH_STORE_HANDLER
<<start>> Start

Figure 5 - ssmmCommandManager

3.7 ssmmBoard

This module is in charge of managing the SSMM physical board. It provides functions for writing, reading and controlling the board. It's modelled as a protected resource. It provides public function for writing, reading and controlling the board.

4 Performing Schedulability Analysis

4.1 Defining the HRT Attributes Values

In the following we describe the criteria we have applied to define such values:

- All worst-case execution times (WCETs) have been calculated using the computational times of the real implementation of the SSMM system in C language, on the Virtuoso Operating System.
- Sporadic object's minimum arrival times (M.A.T.) have been calculated, based on system assumptions.
- The period of the "ssmmHk" cyclic and deadlines for cyclic tasks have been derived from the system requirements.

4.2 Schedulability Analysis Results

All the tasks are schedulable. Below is a table summarizing the results.

Task Name	Type	Wcet (ms)	Deadline (ms)	Period or M.A.T. (ms)	Priority	Blocking (ms)	Utilization (%)	Response (ms)
SsmmHk	Cyclic	0.01208	30.0	1000.0	1	0.00412	1.2080E-3	0.0162
InitSsmmHandler	Sporadic	0.02	40.0	8.61E7	2	0.00412	2.3228E-8	0.0362
ClearSsmmStoreHandler	Sporadic	7.6908	41.0	8.62E7	3	0.00412	8.9220E-6	7.727
Set&SwitchHandler	Sporadic	3.84792	42.0	8.63E7	4	0.00412	4.4587E-6	11.57492
StoreSpRouter	Sporadic	4.10944	45..0	50.0	5	0.016	8.2188	15.69624
SsmmStore1	Sporadic	13.97216	50.0	50.0	6	0.016	27.94432	29.6684
SsmmStore2	Sporadic	13.97216	60.0	50.0	7	0.00412	27.94432	43.628679
CmdReadPageManager	Sporadic	0.0308	210937.0	210937.0	8	0.00412	1.4601E-6	43.65948
SignalDumpManager	Sporadic	102500	8640000.0	8640000.0	9	0.00412	1.1863425	285638.21
SsmmSelfTestHandler	Sporadic	3.00E6	8.64E7	8.64E7	10	0.0	34.722222	8929794.0

Relevant properties derived from the schedulability analysis are presented in the "Utilization Based Analysis" chart (Figure 6), showing the utilization percentages for each task.

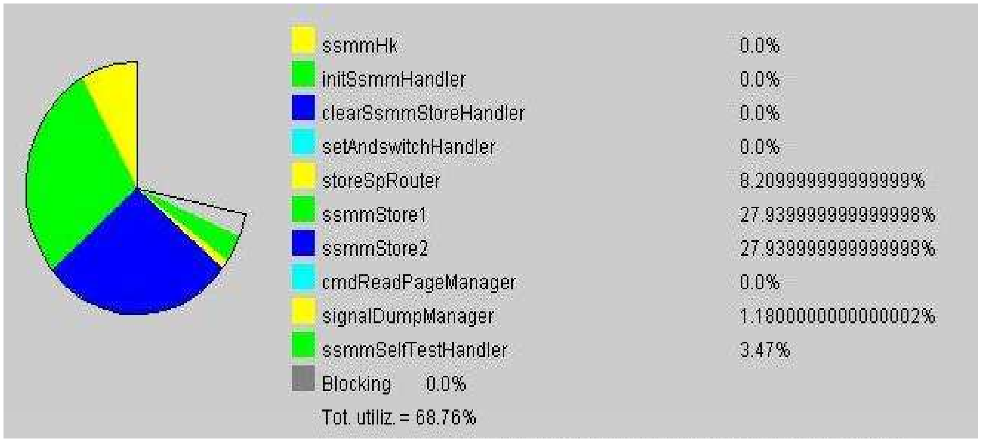

Figure 6 - Utilization based analysis

The system's total utilization is 68.65 %. The greatest part of the computational time is assigned to the "SsmmStorage", which consists of three tasks: storeSpRouter, ssmmStore1 and ssmmStore2. These sporadic tasks only use 63,97 % of the computational time.

On this basis, we activated the analysis tool several times, with different values for the packet minimum arrival time. The chart in figure 7 describes the results calculated by the analysis tool and shows that the total utilization percentage This value strictly depends on the packet minimum arrival time and rapidly decreases, when increasing the latter.

This demonstrate how schedulability analysis results can be used to perform "what if" analysis for the HRT requirements and attributes, possibly contributing to tune system requirements (e. g. increasing the M.A.T.) or raising the need for code optimisation (e. g. to reduce some WCET).

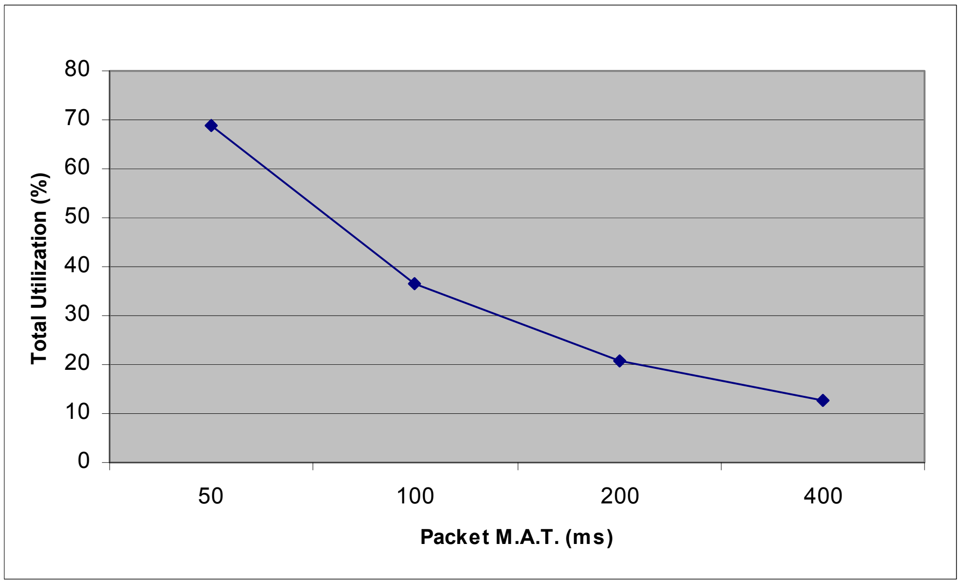

Figure 7 - Total Utilization depending on M.A.T.

5 Evaluation Results

The design of the SSMM software has confirmed the suitability of both the HRT-UML method and the HRT-UML toolset.

5.1 Tasks Identification

The main difference between the SSMM software original UML design and the HRT-UML design is the larger number of tasks in the HRT-UML solution. In fact the original design contains only 6 tasks, while in the HRT-UML model we have 10 among cyclic and sporadic objects, plus a number of 9 protected objects.

This difference is mainly due to the reason that the HRT-UML computational model imposes to split up functionalities among collaborations of the typical HRT tasks, predefined in such a way that their timing behaviour can be predictable.

5.2 Methodological issues

UML is a general language for describing software and also non-software systems, it is not tied to a specific computational model, nor to code generation rules for a specific language, therefore allowing a large flexibility, that can be expressed through many views and different kind of diagrams.

On the other hand it lacks of a specific methodology so that the designer can get bewildered, trying to find the right model elements to use and to formalize its design choices. The case study has proved the following drawbacks:

- Most of the classes are actually singletons and could be more easily considered as simple objects. The designer has to define the meaning of any class diagram, by indicating first level software components, decompositions and associations as use relations. This has confirmed for the SSMM software the need of a topological and de-compositional design approach, instead of a logical classification of software base elements, in the classical bottom-up object-oriented approach. Hierarchical decomposition is addressed by OMG in future UML version 2.0.
- In the original UML design the dynamic behaviour of the SSMM software components is not described, but tasks are identified with textual descriptions separately, and the help of sequence diagrams to describe some scenarios of their collaborations.
- In the UML solution non functional HRT issues are not described. HRT-UML instead offers the designer building blocks already specialized with respect to the real-time behaviour and attributes, so that the designer can early reason about concurrency choices and verify them through system schedulability analysis.

The case study confirms that the main advantage of HRT-UML is its methodological approach to the modelling of HRT systems as a top down decomposition of interacting objects with HRT attributes and following a defined HRT computational model (HRT object types).

5.3 Feedbacks to the SSMM Real Project

No change seems necessary to the original solution, as a result from our re-design activity, according to the software functional requirements. Based on the HRT estimated attribute values, the schedulability analysis was positive.

The major feedback for the SSMM project team comes from the charts built with the support of the analysis tools. Figure 6 describes the utilization analysis results and shows an high value of CPU utilization percentage (68.65%), were most of the time is spent by the SsmmStorage sub-system components, even if SsmmStorage has been implemented accurately, using the most effective solutions. On the other hand figure 7 shows that by increasing the packet minimum arrival time the total utilization percentage decreases rapidly.

The SSMM software is only a part of the software running on the OBDH board and the system RT requirements were not verified at the time the case study was performed. However the analysis results, raised in context of this case study, are an important input for the project: in case the whole software on the OBDH board would be not schedulable at the end, the study has showed how is possible to tune the system. For example specific requirements could be added in order to limit the packet minimum arrival time, by realizing control mechanisms and the handling of the specific case of the arrival of packets too frequent (e.g. lost of packets).

This demonstrates how the adoption of the HRT-UML method, which only requires a smooth transition effort from pure UML, allows the team to acquire a strict control on the timing behaviour of the system. Schedulability constraints are mathematically analysable with results that are unreachable with intuitive or experience-based evaluations.

6 References

[Burns 95] Burns A., Wellings A., HRT-HOOD: A Structured Design Method for Hard Real-Time Ada. Systems, Elsevier, 1995.

[D'Alessandro 02] D'Alessandro M., Mazzini S., Di Natale M., Lipari G., HRT-UML: a Design Method for Hard Real-Time Systems based on the UML Notation, Proceedings of DASIA Conference, 2002.

[Gamma 99] E. Gamma, R. Helm, R. Johnson, and J. Vlissideds [Gang of Four], "Design Patterns: elements of Reusable Object-Oriented Software", Addison-Wesley, 1999.

[HRM 92] HOOD Users's Group, HOOD Reference Manual, Release 3.1.1, 1992

[Mazzini 02] Mazzini S., D'Alessandro M., Di Natale M., Domenici A., Lipari G., Vardanega T., HRT-UML: taking HRT-HOOD into UML, Proceedings of Ada Europe Conference, 2003.

[UMLSP 01] OMG Unified Modelling Language Specification, Version 1.4, September 200