Probabilistically Analyzable Real-Time Systems

D5.3: Project Requirements and Success Criteria
D1.1 Technical Specification: Platform
D2.1 Technical Specification: Software Technology
D3.1 Technical Specification: Probabilistic Timing Method Selection
D3.2: Baseline Integrated Tool-chain

Version 2.0

Document Information

Contract Number	249100
Project Website	www.proartis-project.eu
Contractual Deadline	m6
Dissemination Level	D1.1, D2.1, D3.1 and D5.3 Public D3.2 Restricted*/Public
Nature	D1.1, D2.1, D3.1and D5.3 Report[1] D3.2 Prototype
Author	Francisco J. Cazorla (D5.3), Eduardo Quiñones (D1.1), Tullio Vardanega (D2.1), Liliana Cucu (D3.1), Francisco J. Cazorla (D3.2)
Contributors	All members of all institutions (BSC, RAPITA, UNIPID, INRIA, AFS)
Reviewer	Tullio Vardanega (D5.3), Guillem Bernat (D1.1), Eduardo Quiñones (D2.1), Guillem Bernat (D3.1), Liliana Cucu (D3.2)
Keywords	Project Requirements, Work Package Requirements, Tool Chain Requirements

Notices: *The research leading to these results has received funding from the European Community's Seventh Framework Programme ([FP7/2007-2013] under grant agreement n° 249100.*

[1] All Deliverables marked RE* / PU will be publically available within 6 months of their delivery to the EC

Change Log

Version	Description of Change
v1.0	Initial Draft released to the European Commission
v2.0	Final version

Table of Contents

Acronyms and Abbreviations

AIM	In reference to the AIM alliance formed on October 2, 1991, between ***A**pple*, ***IBM** and **M**otorola* to create a new computing standard based on the PowerPC architecture, AIM designates the first specification for that standard.
APEX	Nickname of the ARINC653 standard. The primary objective of this specification is to define a general-purpose **APEX** (***AP**plication/**EX**ecutive*) interface between the Operating System of an avionics computer resource and the application software.
IMA	An ***I**ntegrated **M**odular **A**vionics* (IMA) system is a particular, in fact the standard, airborne network of real-time, safety-critical computers. This system architecture consists of a number of computing modules capable of supporting numerous applications of differing criticality levels. Adoption of IMA concept permits to achieve an integrated avionics architecture whose application software is portable across an assembly of common hardware modules equipped with a standard software interface. An IMA architecture imposes a vast number of requirements on the underlying hardware and software, here we have only focused in a subset of them. Some requirements that distinguish IMA systems from previous architectures are Incremental Qualification (the ability to not have to recertify unchanged components even though they interact with upgraded components) and Robust Partitioning (an implementation technique chosen for its contribution to arguing the soundness of Incremental Qualification).
OEA	PowerPC ***O**perating **E**nvironment **A**rchitecture*, including the memory management model, supervisor-level registers, and the exception model. These features are not accessible from the user level. This is referred to as Book III of the PowerPC Architecture [PowerISA].
SPE	The ***S**ignal **P**rocessing **E**ngine* is a 64-bit, two-element, single-instruction multiple-data (SIMD) ISA, originally designed to accelerate signal processing applications normally suited to DSP operation. SPE is primarily an extension of Book I but identifies some resources for interrupt handling in Book III-E of the PowerPC Architecture [PowerISA].
Time Determinism	Time Determinism (TD) is achieved when we can determine the state of the system at any time t on the basis of the initial state, inputs and the time cost of the state transition triggered by those inputs. The property of TD is disjoint from that of functional determinism in that a functional deterministic system can also be non time deterministic: consider for example a system whose functionality can be fully described by a finite state machine. Further assume that the transition time from any two states S0 to S1 is a random value in the range [t1:t2]. This system is functionally deterministic in that it will always finish in state S1 from state S0,

	but it is not time deterministic because we cannot determine the exact time at which the transition from S0 to S1 will complete.
Time composability	Given a certain property of each item of a collection, *composability* is attained whenever that property can be determined for each item taken in isolation and does not change when multiple items are brought together. *Time composability* refers to the fact that the execution time of a software component (typically a task, which is the unit of scheduling on a computer system) observed, by analysis or measurement, in isolation, is not affected by the presence of other software components. Ideally, if all system components are individually timing composable then we can replace them at will without this requiring the timing of other components to be re-analysed.
Time compositionality	Given a certain property of each item of a collection, *compositionality* is attained whenever some resulting property can be determined for the whole collection, when the items are brought together to form it, through an economically viable function of the properties determined for each item taken in isolation and for their composition. A system is *time compositional* if we can analyse its overall worst-case timing behaviour using a simple (i.e., economically viable) function of: (i) the independently calculated WCET of its components; and (ii) the (functional/architectural) interaction intended among them.
Time Predictability	Time predictability (TP) is less strong than TD. TP does not require full knowledge about all the events that can occur during execution. Thus, when the timing of an event cannot be determined, pessimistic assumptions are made. This is for example the case with cache accesses, which are considered misses (in the absence of timing anomalies) when the memory addresses are not known in advance. In contrast, TD requires complete knowledge of the system at all points in time.
Timing anomalies	The notion of timing anomaly was introduced in [LuSt99]. The cited work sanctions that a hardware/software component suffers from timing anomalies in its behaviour when a locally faster execution leads to an increase of the execution time of the whole program (or segment of interest). Intuitively, a timing anomaly is a situation where the local worst-case does not entail the global worst-case. For instance, a cache miss – the local worst-case – may result in a shorter execution time than a cache hit, because of scheduling effects [RBW+06].
UISA	PowerPC ***U**ser **I**nstruction **S**et **A**rchitecture* (UISA), including the base user-level instruction set, user-level registers, programming model, data types, and addressing modes. This is referred to as Book I of the PowerPC Architecture [PowerISA].
VEA	PowerPC ***V**irtual **E**nvironment **A**rchitecture*, describing the memory model, cache model, cache control instructions, address aliasing, and related issues. While accessible from the user level,

	these features are intended to be accessed from within library routines provided by the system software. This is referred to as Book II of the PowerPC Architecture [PowerISA].

Executive Summary

This document describes the technical requirements to be accomplished by the PROARTIS Project as well as the criteria that will be used to measure the success of the project. This document includes the following Deliverables due at m6, as described in the Description of Work (DoW):

- Overall Technical Project Requirements and Success Criteria (D5.3)
- Technical Specification: Platform (D1.1)
- Technical Specification: Software Technology (D2.1)
- Technical Specification: Probabilistic Timing Method Selection (D3.1)
- Baseline Integrated Tool-chain: (D3.2)

The scope and contents of these deliverables are so tightly interconnected to one another that we chose to merge them all into a single document, yet keeping each individual deliverable in a separate section for the sake of clarity and ease of reference. Our aim in doing so was to reduce redundancy and increase consistency.

We ordered the presentation of the requirements that emanate from individual deliverables as follows: first the general requirements and success criteria (D5.3), then the per-workpackage requirements (D1.1, D2.1, D3.1) and finally the requirements on the tool chain (D3.2).

Requirements are assigned a unique identification number in this document for future reference along the lifespan of the project. This identification is in the form: *Rn.m* where *n* denotes the originating deliverable and *m* the ordinal within the deliverable.

The identification of the technical requirements of the project is one of the main results of the first 6 months of work. For a complete survey of all the activities carried out during the first six months of project work we refer the reader to deliverable D5.4 (6-Month Project Activity Report).

1 Overall Technical Project Requirements and Success Criteria (D5.3)

As part of the initial period, i.e. the Bootstrap phase of the project (m1:m6), we have identified the requirements baseline for PROARTIS. The deliverable D5.3 “Project Requirements and Success Criteria” specifies the overall technical requirements. By meeting those requirements, PROARTIS will address the overall strategic industrial goals presented in Section B1.1.3 of the DoW, that are: increased performance, reliability and reduced cost (G1); increased productivity (G2); and reduced time to market (G3).

PROARTIS focuses on a development process for Critical Real-Time Embedded Systems (CRTES) in which the analysis and construction of the overall system requires considering the timing behaviour of the individual software components that form that system. To do so, the technical requirements are considered at two levels: (1) the *software component* (SC) level, which contemplates the analysis – in isolation – of the individual SC that constitute the system, and (2) the *system* level[2], which contemplates the analysis of the system as a whole, including the interactions among its constituent components. Such an approach will enable us to better adhere to the IMA[*] standards. To that end, we consider *boundaries* between SC those that match our choice of boundaries between what we will declare as *separately qualified software components*.

The technical requirements identified at the *software component* level during the first six months of the project are:

- (**R53.1**) To attain confidence in the worst-case execution time (WCET) estimations obtained by use of the PROARTIS techniques. This confidence will be measured in the form of probabilities – the desired region of probabilities is 10^{-9} - 10^{-12} – and the confidence in such probabilities.

This requirement can be broken down into several sub-requirements concerning timing analysis, performance and industrial viability:

- (**R53.2**) Timing analysis: (WP3) To develop trustworthy WCET estimation techniques based on probabilistic analysis of the behaviour of (WP1/WP2) the PROARTIS hardware (HW) and software (SW).

- (**R53.3**) Timing analysis: (WP4) To provide trustworthy confidence on the resulting WCET estimations such that the PROARTIS techniques can be used in the context of typical Airbus airborne applications to feed system-level techniques, such as end-to-end response time analysis.

- (**R53.4**) Timing analysis: *Time composability*[*] of individual SC has to be guaranteed in the face of all other SC. The architecture and execution model of the system must ensure that the timing behaviour of SC in isolation is time composable when SC are integrated with other SC to form the system. In that

[2] The term “system” does not refer to the aircraft, car, etc. level, but the *computer system* level, that is one piece of hardware that is colloquially called “a computer” and all the pieces of software that are loaded inside said computer.

[*] All terms marked with an asterisk are defined in the Acronyms section.

manner, individual SC can be developed and analysed independently of the Operating System (OS) and of the other software components. These timing properties are a prerequisite to the analysis techniques developed in WP3.

- (**R53.5**) Timing analysis: Time robustness at the SC level is the property such that small changes in the input set used for timing analysis do not have an inordinate impact on the WCET estimation. Examples of input set data for the timing analysis are small shifts in the code or data placement in memory, the knowledge of whether some memory operations are hits or misses or lack of knowledge on some parameters of the hardware configuration. A PROARTIS goal in this regard is to reduce – as opposed to increase – the level of detail of hardware knowledge needed to perform useful and trustworthy timing analysis.
- (**R53.6**) Performance: Whereas time composability (**R53.4**) facilitates incremental qualification, it may also introduce performance degradation as it may renounce optimizations or accelerations resulting from interaction with and interference from other components. We will study the extent of this negative effect on performance and will strive to minimize it. In particular, the WCET estimates obtained by using the PROARTIS approach must permit industrial software applications to confidently attain a level of performance not inferior to current practice (expressed, for example, in terms of "safe" CPU load) and to permit the adoption of more advanced (thus less predictable) and powerful hardware.
- (**R53.7**) Industrial viability: The PROARTIS approach shall be deemed "industrially viable" if it can be deployed within levels of time and cost no higher than with current practice, whilst achieving the above improvements.

The technical requirements identified at the *system level* during the first six months of the project are:

- (**R53.8**) To enable *incremental qualification* of the system.

  Rationale: Incremental qualification is one of the distinctive properties expected of IMA-compliant systems. In that manner, system components can be developed and verified in isolation in such a way that they can be changed, added or upgraded with minimal, ideally negligible, effect on the timing behaviour of the other system components.

To this end, PROARTIS solutions are required:

- (**R53.9**) To ensure *time compositionality** at the system level**,** in a static and economically viable manner.

  Rationale: Time compositionality ensures that the timing behaviour of the whole system is determined, in a static and economically viable manner, by the timing behaviour of the application components (SC) that compose the system and by their planned and allowable interaction. At integration time, we must possess a composition function that takes in input the timing properties of SC and their interaction and yields time compositional outcomes.

That is, a system is time compositional if we can analyse its worst-case timing behaviour using a simple (i.e., economically viable) function of: (i) the independently calculated WCET of its components; and (ii) the (functional/architectural) interaction intended among them. If all system components are individually time composable then we can replace them at will without this requiring the timing of other components to be re-analysed.

- (**R53.10**) To ensure time robustness at the system level: the system resulting from the PROARTIS project should be timing robust. A timing compositional system is robust in that changes in the timing behaviour of individual components only have a component-local effect on the timing behaviour of the system.

By meeting these requirements at the application level and at the system level, PROARTIS will allow incremental (and thus hierarchical) development, shortening verification activities during the development cycle, to enable practices such as co-design, continuous integration, etc., and from release to release, to reduce the cost of requalification.

1.1 PROARTIS approach

The PROARTIS project focuses on the time behaviour of IMA-compliant systems. The time analysis of a system can be performed either *statically* or *empirically*.

Static Analysis is the recommended practice for the timing analysis of CRTES. Static analysis methods require the underlying system to be time deterministic (TD). In a time deterministic system, for a given input set, the sequence of events that will happen in the system is fully known, and so is the time at which the output will be produced. This form of analysis obviously needs a very accurate and detailed model of the system. The safety and tightness of timing analysis depend directly on the accuracy of the available system model, which in turn depends on the accuracy of information we can obtain about its actual functioning and timing.

Static Analysis can be also applied to time predictable (TP) systems, in which the timing of the events that can occur during execution is not known in advance, but can be bounded with acceptable confidence. When the timing of an event cannot be determined, pessimistic assumptions must be made about its occurrence. This is, for example, the case with cache accesses, which must be considered misses when the memory address issued is not known beforehand; hence when the time of their occurrence cannot be predetermined. (Note that this assumption only holds in the absence of timing anomalies[*].) In contrast to TP, TD requires complete knowledge of the system. What really matters of the knowledge required is not its quantity but the cost of acquiring it. Modelling a TD system requires more information than a TP system, possibly much more. For TP it is important to either know when activities start and end (which we refer to as the *strongest* notion of TP) or be able to fix regions for the start and end times (which we refer to as a *relaxed* notion of TP, where an activity starts in the time region $[t_1:t_2]$ and finishes in the time region $[t_3:t_4]$ with $t_1 \le t_2 \le t_3 \le t_4$).

With the increasing complexity of CRTES (in terms of both hardware and software components), the level of knowledge needed to attain the performance required for new-generation systems, as well as the time effort, cost and complexity required to acquire and comprehend all those needed details of the system become plainly unaffordable. The central hypothesis of PROARTIS is that new advanced hardware features can be used and analysed effectively in embedded real-time systems with designs that provide time randomized behaviour. This shift will enable probabilistic analysis techniques that can be used effectively in arguments of verification of CRTES, proving that the probability of pathological execution times is negligible and therefore tighter bounds can be safely obtained.

The PROARTIS approach to accomplish with the afore-mentioned requirements is:

- (**R53.11**) To identify the HW and SW *sources of dependency of the WCET of SC on the execution history*.

Dependency on (or sensitivity to) execution history may prevent the use of probabilistic analysis techniques, as we lose the independence between events. Examples of such dependency are:

- At hardware level, the basic functioning of the cache is intrinsically dependent on execution history. Its behaviour therefore is a source of such dependency. The probability of hit/miss of a given access depends on whether previous accesses hit or miss in the cache. A small shift in the code/data in memory may cause previous accesses to go to different cache sets, which may have large impact on the WCET estimations.
- At software level, the WCET estimation that can be obtained for the execution of the code of a system call may depend on the actual time of the call and on what applications executed (and for how long) since the previous call.

Along the lifespan of the PROARTIS project, we will follow an iterative process in which we will identify the sources of dependency and try to defeat them by introducing **randomization** in their operation. As depicted in Figure 1, the iterative approach followed in PROARTIS will be as follows:.

- In WP1/WP2 we will capture and attack, at both hardware and software levels, the principal sources of dependency on execution history that do not permit to exploit the performance benefits of advance hardware features, also ensuring that the hypotheses made in the approach to probabilistic analysis devised in WP3 hold at all times.
- WP3 will provide the means to verify that the requirements on the behaviour of the system for it to permit the application of probabilistic analysis hold.
- In WP4 we will assess the applicability of the methods, tools, techniques and arguments developed in WP1, WP2 and WP3 to the case of certified avionics.

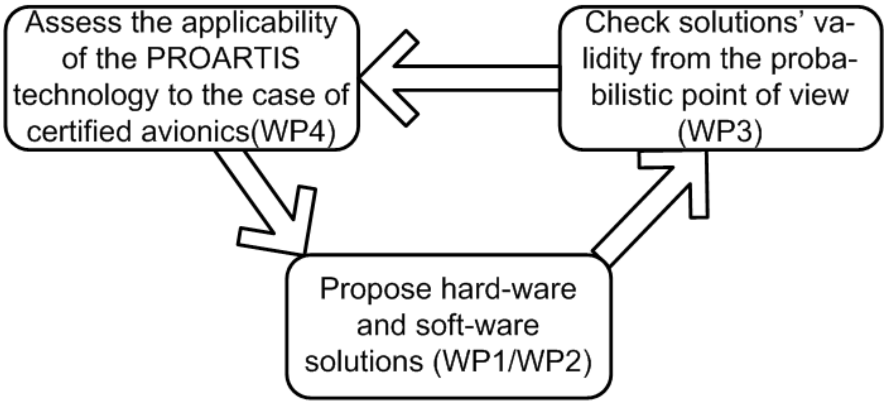

Assess the applicability of the PROARTIS technology to the case of certified avionics(WP4)
Check solutions' va-lidity from the proba-bilistic point of view (WP3)
Propose hard-ware and soft-ware solutions (WP1/WP2)

Figure 1. Iterative process to be followed in the lifetime of the project

The "angle of attack" taken by PROARTIS is to develop an experimental execution platform (HW and SW up to and including the middleware) whose timing behaviour has as low dependency as possible on execution history. The investigation approach taken in the project will be incremental: for the HW level we will begin by addressing the cache as one of the principal sources of dependency on execution history; WP2 will ensure that the OS keeps the properties required by the HW and SW so that the system can be analysed with the probabilistic analysis techniques devised in WP3. WP2 may extend the randomization principle to the operation of the OS by identifying and redesigning the data structures and run-time algorithms whose timing behaviour depends on execution history to the point of impacting on the WCET predictions significantly.

The cost in HW/SW development and/or procurement, and the increment in execution time that may be required to attain time composability will be evaluated. For example, the ideal conceptual cache we are after is one with no dependence on execution history at all. In this cache, on every access we may completely flush the cache and reload from memory as many data as the size of the cache from random positions. The probability of hit of a memory access in such a cache system is given by: cache size/memory size (N/M). Interestingly, this ratio does not depend on execution history. The HW cost of such a system is not high, but the resulting performance may be unacceptably poor for small caches or large memories. We must thus pragmatically relax the extent of independence we want on execution history. For example, a cache in which on every access we evict a cache line from a random position has the following property: the probability of hit of a given memory operation for which the corresponding cache line *cl* was already fetched by a previous memory operation, only depends on the number (*K*) of memory accesses intervened since *cl* was last fetched and that may have potentially evicted *cl*. The determination of *K* is where we depend on execution history. *K* reflects the *reuse distance*.

In the final phase of the project (*Multicore phase*), we will look at multicore architectures. At that point, in addition to dependency on execution history, we will also focus on the dependency of the WCET on the *environment*. We have to provide solutions to allow tasks to run simultaneously in a multicore architecture, in a manner that permits tasks to preserve their timing properties.

In that phase of the project, once we understand how the PROARTIS approach behaves in a timing-anomaly free architecture, we will also investigate how the PROARTIS approach works in the presence of timing anomalies. This investigation will be conducted in a best-effort manner, going as far as the residual resources will permit.

1.2 Taxonomy of Analysis Methods

Static analysis requires HW and SW to have deterministic behaviour. PROARTIS represents a shift in the design of HW/SW architectures in that it requires them to have randomization properties, with no negative effects on the functionality but positive effects on the elimination of timing anomalies and other dependences on execution history. Table I breaks down all possible pairs of time analysis approach and the type of HW/SW platform that best fits them.

Table I. Breakdown of HW/SW behaviour and type of timing analysis

	Measurement Based Timing Analysis		**Static Timing Analysis**	
	Conventional	Probabilistic	Conventional	Probabilistic
Deterministic HW/SW	Profiling	State-of-the-art (Copulas)	State-of-the-art	N/A
Randomized HW/SW	Profiling	PROARTIS	Extremely pessimistic	PROARTIS

In Table I we consider two fundamentally alternative types of HW/SW: deterministic (as in the state-of-the-art) and randomized (as in the PROARTIS proposal).

- The deterministic HW/SW set captures the current HW/SW architectures (at least those intended for CRTES) whose purpose is to be as time deterministic as possible.
- The randomized HW/SW set corresponds to the type of HW/SW architecture sought by PROARTIS to facilitate the development of the mathematical and models needed to perform probabilistic and statistical analysis of the system timing behaviour.

The timing analysis of an application can be performed:

- By means of measurements on the real platform, which depend on how representative the actual tests are.
- By means of static measurements on models of the real platform, which in turn depend on the accuracy of the model.

The following combinations of analysis and HW/SW behaviour are possible:

- Conventional *Measurement-Based Timing Analysis (MBTA)*, measuring a piece of software end to end can be performed on deterministic HW/SW.The number of tests required to feed a trustworthy timing analysis can be extremely high because even small changes in the input set can result in large performance variations. The approach of commercial tools like RapiTime from Rapita Systems aims to reduce the number of tests required to achieve a WCET result by allowing long events from one test to be combined with other test results. Nevertheless, systematic pathological cases, which lead to large increments in WCET, can still be hard to detect. MBTA analysis can be performed on randomized HW/SW as well. However, since this type of HW/SW makes pathological cases non-systematic but does not remove them, we can expect also high (i.e., not tight) WCET estimations.
- *Measurement-Based Probabilistic Timing Analysis* (MBPTA) has routinely been performed on deterministic HW/SW. MBPTA is based on the detection of independent events to obtain tighter and safer WCET estimations, but its benefits are very limited due to the history-dependent nature of such HW/SW. PROARTIS pursues MBPTA on randomized HW/SW, where assumptions on history-independent events can be made and therefore tight probabilistic WCET

estimations can be obtained. The measurement based process also enables *Statistical analysis* to be performed on the PROARTIS architecture.

- *Static Timing Analysis* (STA) is routinely performed on deterministic HW/SW. Unfortunately STA accuracy has very high dependence on the amount and quality of the information available on the model and on the quality and accuracy of the analysis technique itself. Thus, as systems and applications become more complex, STA rapidly loses tightness and accuracy, thus becoming more pessimistic, because the cost to acquire all of the information needed to perform accurate WCET estimations becomes increasingly unaffordable. STA analysis could be performed on top of randomized HW/SW. However, due to the safe nature of STA, pessimistic assumptions should be taken for all those events arising from a randomizing execution environment, thus leading to extremely pessimistic WCET estimations.
- PROARTIS pursues *Static Probabilistic Timing Analysis* (SPTA) on randomizing HW/SW where safe assumptions do not imply assuming the worst possible outcome but using each possible outcome with their corresponding probabilities to provide safe and tight probabilistic WCET estimations. As a consequence, the model required by SPTA is simpler than the one required by SPTA, as the former does not have high dependency on execution history. Finally, SPTA on deterministic HW/SW does not make sense since it relies on some underlying properties unavailable in such type of HW/SW.

1.3 Per WP-requirements and inter-WP requirements

In addition to these general technical requirements, other per-workpackage and inter-workpackage requirements have been identified during the bootstrap phase of the project. All of them are detailed throughout the description of the different technical deliverables in this document (D1.1, D2.1, D3.1 and D3.2).

We have also identified the requirements that the different workpackages have set on the PROARTIS tool chain. A complete description of the tool chain is given in D3.2.

2 Technical Specification: Platform (D1.1)

Deliverable 1.1 *Technical Specification: Platform* aims to determine the requirements of the baseline platform infrastructure suited for the PROARTIS research needs. D1.1 has to be capable of meeting the requirements emanating from WP4 for the development of the avionics case study and the other requirements coming from D2.1, D3.1 and D3.2. The baseline platform infrastructure is comprised of two levels: the *hardware level*, which comprises the processor architecture, and the *hardware-dependent software level*, which comprises the compiler, the linker and the run-time libraries concerning memory allocation.

It is worth noting that, whereas WP1 and WP2 both focus part of their research at the software level, there is a clear boundary between their respective responsibilities: run-time libraries in the context of WP1 are understood to be exclusively applied to sequential (single-threaded) execution; conversely, WP2 addresses those run-time libraries needed to support concurrent (multi-threaded) execution.

This deliverable summarizes the work done in Task 1.1 *Study of hardware requirements and hardware-dependent software requirements*. All the objectives defined in this task for the m1:m6 period were achieved.

2.1 Specification of Research Objectives as Understood at Milestone 1

WP1 has three main research objectives, fully in line with the proceedings of the requirements capture phase conducted in the first 6-month period described in deliverable D5.3. These objectives can be described as follows: WP1 will design, prototype and evaluate hardware techniques (i.e., processor architecture) and hardware-dependent software techniques (i.e., compilers, linkers and run-time libraries that affect the memory management) that:

- **O1.1.** Reduce and possibly annihilate the dependence of application timing on execution history.
- **O1.2.** Provide a level of performance that allows industry to afford the verification and validation cost impact arising from the adoption of more advanced and powerful system solutions.
- **O1.3.** Ensure that the timing behaviour of the hardware and hardware-dependent software techniques developed in WP1 fits the assumptions made by analysis techniques developed in WP3, such that trustworthy confidence on the resulting WCET estimations can be used in the context of typical Airbus airborne applications (WP4).

The strategy adopted in WP1 for the evaluation of the results obtained against the objectives will be incremental and progress from laboratory experiments to industrial case studies.

2.1.1 Rationale

Objective **O1.1** covers one of the key challenges arisen in requirement **R53.11**: to identify and defeat the sources of dependency of the WCET bounds on the execution history.

This deliverable identifies those high performance hardware components that, despite their advantageous performance benefits to cope with current and future hard real-time system needs, require massive and costly investigation efforts to permit tight time analysis with current WCET analysis approaches. This deliverable also analyses the impact that the hardware-dependent software has on the execution history of the hardware components previously highlighted. Looking for example at the cache, the set of memory addresses that a program accesses depends on how objects (instructions and data) are organized in memory. Hence, different organizations of objects in memory lead to different timing behaviours.

Objective **O1.2**, which covers requirement **R53.6** defined in deliverable D5.3, aims to not renounce the performance benefits of advanced hardware features due to time analyzability issues.

Finally, objective **O1.3** ensures that the assumptions made by the timing analysis method are maintained by the new hardware and hardware dependent software techniques developed in PROARTIS. These assumptions are still not fully defined at the time of this writing; however, it is clear that objective O1.3 is crucial for the **R53.3** requirement defined by deliverable D5.3, which will provide trustworthy confidence on the resulting WCET estimations therefore backing the use of PROARTIS techniques in the context of typical Airbus airborne applications (WP4) to feed system-level techniques, such as end-to-end response time analysis.

To sum up, the purpose of WP1 is to make the cost of acquiring the information required to perform timing analysis affordable. The strategy of WP1 consists of injecting randomness into those advance hardware components whose timing behaviour is execution time sensitive (**O1.1**). By doing so, we expect that the performance benefits bought by these components can be exploited by CRTES (**O1.2**), while still attaining the required level of confidence on the resulting WCET estimation (**O1.3**).

2.1.2 WP1 Top-level Requirements

Based on the earlier discussion we can identify four requirements to be satisfied by the other project work packages so as to ensure a successful development of the PROARTIS research:

- (**R11.1**) The software techniques developed in WP2 have to maintain (or even augment) the reduction on the execution history dependency achieved by WP1 techniques. Such a requirement will be covered by objective **O2.1** defined in deliverable D2.1.
- (**R11.2**) WP3 has to provide mechanisms to check if the independency from sensitivity to execution history achieved by the techniques developed in WP1 are sufficient to comply with and preserve the assumptions made by the timing analysis.
  Rationale: See objective O1.3 defined in section 2.1.
- (**R11.3**) WP4 has to establish whether WP1 hardware and hardware-dependent software designs are workable in the context of certified avionics.

- (**R11.4**) WP4 has to establish whether WP1 hardware and hardware-dependent software designs achieve the desired/expected computational performance.
  Rationale: See objective O1.2 defined in section 2.1.
- (**R11.5**) The avionics demonstrators to be developed in WP4 shall be capable of determining whether the hardware design solutions proposed in WP1 are practicable, i.e., economically implementable and performance-wise viable in the context of certified avionics.
  Rationale: This requirement reflects the strong industrial focus of PROARTIS.

2.2 PROARTIS Platform

This deliverable defines the baseline platform needed to develop the PROARTIS research. During the first six months of the project, we have identified those components at hardware and hardware-dependent software level that provide high performance level at the price of being the main sources of dependency on execution history:

- At the hardware level, we have identified the memory hierarchy and the interconnection network. The timing behaviour of both components is *sensitive to execution history,* i.e., the response time of both components at any given point in time depends on previous accesses made by the same application or different applications.
- At the hardware-dependent software level, we focus on three software components: the compiler, the linker and the run-time libraries serving sequential (single-threaded) execution that may affect the behaviour of the memory hierarchy.

During the lifetime of the project other components may become of interest for the PROARTIS research. This is the case, for example, of the branch predictor. Our initial studies have concluded that, although the branch predictor is an important source of execution history dependency, its correct behaviour highly depends on this dependency. Thus, the elimination of the execution history dependency would result in a performance loss, removing the benefits provided by the branch predictor altogether. However, future PROARTIS results may cause us to focus again on the branch predictor.

In general, the PROARTIS research shall operate within the following perimeter:

- (**R11.6**) The components of interest to PROARTIS shall be such that their timing behaviour depends on the execution history and the reduction of that dependency does not cause unacceptable performance degradation (in terms of WCET analysis) and incorrect functional behaviour.
  Rationale: See objective O1.2 defined in section 2.1.
- (**R11.7**) PROARTIS solutions will be independent of the specific processor architecture in use.
  *Rationale: We highlight the importance of requirement **R11.7**: whereas the PowerPC instruction set architecture has been selected in the PROARTIS platform baseline to facilitate the development of the industrial case study (see D3.2), all hardware solutions developed in the project will be independent of any specific ISA. This is of paramount importance to export the PROARTIS technology to other embedded domains, such as space, automotive, construction machinery, etc., in which other processor architectures are used.*

2.3 Hardware Level

This section analyses the sources that make the timing behaviour of the different components of the memory hierarchy and the interconnection network sensitive to execution history.

2.3.1 Memory Hierarchy

Despite its well known performance benefits, the memory hierarchy poses a serious challenge to timing analysis because the response latency of a memory request depends on which level of the memory hierarchy the required datum actually resides. It is important to notice that the access to the first level in the hierarchy (the register file), does not complicate timing analysis at all, as its access is explicitly encoded in the instruction through a unique register identifier. All program objects however reside in the last (outermost) level of the memory hierarchy, which in our case is the main memory[3]. Therefore, in order to perform timing analysis of access to the memory hierarchy, we need to know at any point in time in which level, i.e., whether any of the different cache levels or the main memory, every memory object (instruction and data) accessed by the application resides.

The level in which a memory object resides depends on the execution history. That is, because of the temporal and spatial locality characteristics of the memory hierarchy, a memory object resides in a given memory level if: (1) the memory object has already been fetched; and (2) the memory object has not since been evicted by another request. Thus, in order to perform timing analysis of access to the memory hierarchy, current analysis methods must keep track of all memory accesses, of the absolute order among them, as well as of the granularity used to fetch them to determine if an object has been evicted.

The use of memory systems such as DDRx SDRAM as main memory components can introduce variations in the access latency due to execution history as well. Although this type of memory is commonly used in high performance computer systems, especially in multicore processors, the current trend is to introduce them also in CRTES as performance requirements increase [Wo07]. These types of memories suffer from several sources of timing variability due to: (1) the SDRAM device; and (2) the memory controller.

The following two sections discuss the sources of execution history dependence in both memory hierarchy components, the cache and the main memory.

Caches

Caches are a fundamental hardware component to high-performance computing in that they reduce the long latency access to main memory by storing, close to the processor, the most recently used data. In order to reduce the traffic overhead between the main memory and the cache consecutive memory objects are grouped into memory blocks called *cache lines*, The least significant bits of the address of the memory object, called *offset*, determine the position of the object inside the cache line.

The position in which a cache line is placed into the cache, the *cache set*, is determined by the *mapping function*, which is a hash function map that uses certain

[3] High performance systems typically have higher levels of memory hierarchy, such as hard disks. However, due to their long latency access, such devices are not commonly used in safety critical real-time systems.

bits of the memory address called *index*. Because different cache lines can collide into the same cache set, caches consist of a given number of lines (typically the same number per set) called *ways*. The size of each cache set is called the ***k****-associativity* of the cache. By doing this, the cache lines that collide into one and the same cache set are distributed among *k* ways. Thus, a cache memory consists of K·S cache lines arranged in S sets (conceptually rows) and K ways (conceptually columns).

The way in which a cache line is placed into a cache set is determined by the *cache replacement policy.* The replacement policy decides, once all ways of a cache set are occupied, which cache line is evicted to make room for the new cache line. Numerous replacement policies have been proposed and implemented for both high performance and embedded processors. Most of them are based on information about how frequently cache lines have been accessed. The most common policies are *Least Recently Used (LRU), pseudo-LRU, First-In First-Out (FIFO)* and *random*.

The timing behaviour of a cache is given by the mapping function and replacement policy. Thus, in order to analyse the timing behaviour of a cache using current approaches we must determine the state of the cache at any point in time. The amount of information required to do this is the following:

i. *The complete list of memory addresses accessed by the program.* The memory address (index) and the mapping function determine the cache set in which the cache line resides. From this information, *information about the access frequency of every memory address* can be derived. Given a cache set, a previously accessed cache line resides inside it depending on: (1) how frequently this cache line has been accessed; and (2) how many different accessed cache lines collide into one that the same cache set.

ii. *Detailed information about the cache configuration.* Different implementations of the mapping functions and replacement policies will lead to different timing behaviours. For example, in case of the LRU replacement policy, we must maintain, for each cache set, an ordered list indicating when the last access of each cache line occurs. Every time a cache line is accessed we must reorder the list again. In case of LRU the list will be ordered such that the first element of the list is the least recently used.

Unfortunately, not all memory addresses of a program are statically known. In that situation, current analysis approaches make pessimistic assumptions. For example, whenever a memory address is not known, the timing analysis considers the access as a cache miss (in the absence of timing anomalies). Moreover, we can neither update the frequency access information in case the requested cache line resides in cache, nor know which cache line has been evicted in case the requested cache line had to be fetched to cache. Hence, all the lines of the cache have to be degraded 1 position towards the LRU, which rapidly reduces the benefits of the cache. In particular, K consecutive accesses to unknown addresses will require assuming that all cache contents have been evicted because those K accesses could have evicted all entries from any cache set.

Main Memory

JEDEC-compliant DDRx SDRAM memory systems [JEDEC08] (DDR, DDR2 and DDR3) are probably the most common off-chip memories used in current computer systems, and the current trend is to introduce them also in CRTES as performance requirements increase. They are composed of an SDRAM memory controller and an off-chip SDRAM memory device.

A *memory device* consists of multiple banks that can be accessed independently. The *SDRAM-access protocol* defines strict timing constraints [Ja08] that dictate the minimum time interval between different combinations of consecutive SDRAM commands, according to memory characteristics specified by the JEDEC industrial standard and provided by memory vendors. This notwithstanding, the time an access to the memory device takes depends on the previously executed commands (i.e., it depends on the execution history).

In order to maintain data integrity inside the memory cells, data values must be periodically read out and restored to the full voltage level, otherwise the memory would be unable to read the values again in the future. This operation is called refresh and it is performed through a refresh (REF) command.

The *memory controller* is the interface between the processor and the off-chip SDRAM. It handles the memory requests coming from the processor and translates them into a sequence of commands that are sent to the different banks, guaranteeing that the timing constraints of the memory system are respected.

The timing behaviour of the memory system depends on the SDRAM-access protocol and the memory controller.

- Regarding the SDRAM-access protocol, different combinations of SDRAM commands lead to different timings, making the execution time of a memory request dependent on the sequence of commands issued by previous requests.
- Regarding the memory controller the major causes of time variability are: the Row-Buffer Management policy, the Address-Mapping Scheme and the Arbitration policy [Ja08].

Hence, in order to analyse the timing behaviour of main memory using current approaches it is mandatory to determine the state of the memory system, considering both the memory controller and the memory device, at any point in time, in terms of SDRAM commands issued. The amount of information required to do this is the following:

i. *The sequences of SDRAM commands issued.* The latency of a memory request depends on the SDRAM commands that compose the request. At the same time, the sequence of SDRAM commands depends on the state of the memory system.

ii. *The precise points in the program that SDRAM refreshes will affect.* As noted above, during the time the REF command is issued, no other command can be sent, thereby increasing the execution time of those other requests. The impact that a REF command can have depends on the point of the program in which the refresh occurs. Unfortunately, the exact instant in which the refresh operation occurs cannot be statically determined because it depends on when the application starts.

iii. *Detailed information about the memory controller configurations.* The memory controller is highly configurable, making it possible to define the row buffer policy, the SDRAM refresh periodicity, the arbitration policy used, etc.

Different memory controller configurations will lead to different execution times.

2.3.2 Interconnection Networks

The interconnection network is a key component in current embedded processor architectures. It is in charge of distributing the requests coming from the different cores to the various processor components such as other cores, I/O devices, caches, memory systems, etc. Thus, the interconnection networks play a central role in determining the overall performance. If the network cannot provide adequate performance and predictability for a particular application, nodes will frequently be forced to wait for data to arrive.

Many different network topologies have been used in CRTES, with the objective to provide the performance required by the system, while attaining some levels of predictability:

- FlexRay is an automotive bus standard designed to allow microcontrollers and devices to communicate with one another within a vehicle [FLEX].
- Advanced Microcontroller Bus Architecture (AMBA) is a bus specification that defines an on-chip communications standard for designing high-performance embedded microcontrollers [AMBA].
- SAFEbus backplane is an avionics bus implementation from Honeywell based on the ARINC 659 standard [SAFE].

The key architectural component of the interconnection network is the *arbitration selection policy*, which selects the order in which the requests (from the same application or from different applications) are processed, potentially delaying the issue of other requests. Thus, the amount of information needed to determine the timing behaviour of an interconnection network is the following:

i. *The exact time at which each request accesses the interconnection network.* The state of the interconnection network at any point in time depends on the requests pending to access it.

ii. *Detailed information about the interconnection network configuration.* The delay suffered by every request depends on the arbitration policy used, e.g. round robin, time division multiplexed access, prioritization, etc.

2.4 Hardware-Dependent Software Level

In the m1:m6 period we have also identified the major sources of dependency on the execution history at the hardware-dependent software level..

As noted in section 1.2, the state of the memory hierarchy is determined by the memory addresses accessed by the program, i.e. the *memory placement*. The memory placement defines the memory position of the different memory objects that form an executable. The memory position in which each object resides depends on the hardware-dependent software level; concretely the compiler/linker and the memory allocation run-time libraries. Thus, the hardware-dependent software is responsible for determining the memory placement of the program.

2.4.1 Compiler and Linker

The compiler and the linker are the software components in charge of generating the executable file to be executed in the *host machine*, understanding host machine as the hardware platform and the operating system in which the program will run. In that file we can identify three main segments, each of which is allocated to a different memory position:

- *Code Segment*. This part contains the instruction code. It is commonly the case that functions are allocated in consecutive memory positions, in the same order as given by the programmer. The position in which instructions reside in the code segment and thus in memory do not change with the execution of a program, hence it is independent from which part of the program functions are called.
- *Static Data Segment.* This part contains the global and constant data structures whose size does not change with the program execution and can be accessed at any point of the program. Similarly to the code segment, the position in which the static data resides in the static data segment does not change with the execution of a program, as it is independent from which part of the program data is accessed.
- *Dynamic Data Segment.* This part contains the stack and the heap data structures, whose size may change with program execution and can be accessible only by certain parts of the program. The position in which elements reside inside the dynamic data segment may vary with the execution of the program, and it depends on the control flow path taken by the execution.

The compiler and the linker are the software components in charge of determining the position in which the elements that form the code and the static data segments are allocated into memory, as well as generating the code to manage the stack data structure contained into the dynamic data segment.

Therefore, the compiler and the linker determine the memory placement of the application that will affect the behaviour of the memory hierarchy. Different memory placements will lead to different memory locations of objects, potentially conflicting into cache. For example, if two functions collide into the same cache lines, they should not be called inside a loop one after the other, as one function could evict the other, eliminating the performance benefits of the cache. In [QBB+09] we analysed the impact that different memory placements have on the instruction cache hit rate.

2.4.2 Memory Allocation Run-time Libraries

The run-time libraries concerned with memory allocation are in charge of managing the new data structures allocated by the program into the heap. When the program requires extra memory for allocating new data structures, the memory allocator takes a chunk of free memory from the heap structure. This causes the position in which this new object is allocated to depend on the state of the heap.

Therefore, the memory position in which objects are allocated into the heap depends on the order in which these objects have been requested. Such a dependency makes it extremely difficult to determine the memory placement of the heap and thus the position that these objects will have in the memory hierarchy, as the same object can be allocated into different memory positions depending on previous requests.

It is important to clearly define the boundaries between WP1 and WP2. Hence, the research carried out by WP1 in the context of run-time libraries is exclusively focused on sequential (single-threaded) execution, while WP2 focuses on concurrent (multi-threaded) execution.

2.4.3 Hardware Level Dependency

To sum up, the effectiveness of the memory hierarchy depends on the hardware-dependent software, as it is the one in charge of determining the memory placement. Such a dependency on the memory placement further complicates the timing analysis of the memory hierarchy. That is, if the memory placement of a program changes slightly, for example by introducing a new function, the timing behaviour of the other functions may change dramatically, potentially invalidating the results of the previous WCET analysis. Moreover, the use of heap structures complicates the analysis, which needs to determine the “worst” position in which different memory objects can reside.

The PROARTIS approach will attack such a dependency by injecting randomness in the position at which objects are allocated in the different application segments. For example, by randomly allocating objects that reside in the heap one could remove the dependency on the state of the heap.

It is interesting to notice that the close relationship in place between the memory hierarchy and the hardware-dependent software may avoid the need to inject randomness at both hardware and hardware-dependent software level. For example, one could envision a compiler technique that randomly allocates a variable in the heap every time that variable is accessed. The timing behaviour of accesses to this variable in the cache would be equivalent to those of a fully random cache, in which every time the variable is accessed it is randomly placed into the cache. This notion results in the following requirement:

(**R11.8**) WP1 has to identify the implications that the injection randomness at one level, i.e. hardware or hardware-dependent software level, has on the other level.
Rationale: Such a requirement will study the interaction between the hardware and hardware-dependent software levels when injecting randomness, answering the following questions: Is randomization required at both levels? Are they complementary? Do both levels bother each other? Note that these questions are fully aligned with the strong industrial focus of PROARTIS, as the R11.8 will try to select at which level randomness can be more effective taking into account the industrial viability of the technique.

2.5 Specific Avionics Application Requirements

This section analyses the extra components that the baseline platform requires to accomplish with the WP4 *Case Studies* requirements. Other components, such as the floating point unit, or specific register implementations are discussed in the deliverable D3.2.

Memory Management Unit (MMU)

In a multiprogramming environment, such as the one considered in PROARTIS where several programs with different criticality levels may run concurrently, the operating system and the processor architecture have to provide mechanisms to limit memory accesses to protect the memory space of a program (including instructions and data) without having to swap the program to, for example a disk, on a context switch.

The *memory management unit* (MMU) is the component responsible for handling the memory requests issued by the cores. Among its various functions, a most important one is to translate virtual addresses into physical addresses (i.e., a.k.a. virtual memory management) for the purpose of providing memory protection, i.e. to prevent applications from tampering with each other's data.

One solution to do so is to divide the physical memory into blocks, called *page frames*, and allocate independent subsets of pages to the different processes, such that processes have the rights to access their own page frames only. These rights are managed by the Operating System. The *translation lookaside buffer* (TLB) is the processor hardware component of the MMU used to speed up the virtual address translation. A TLB has a fixed number of slots that contain a subset of the page table entries, which map virtual addresses to physical addresses. When a page frame is requested by the program and it is not present into the TLB, a *TLB miss* occurs, and the new page is brought from the page table, generally loaded in memory, which holds trace of where the virtual pages are stored in the physical memory. This makes the latency of the MMU access depend on the state of the TLB, which in essence is a cache-like structure.

Therefore, the timing behaviour of the MMU is execution history dependent, because the access latency to the MMU depends on the state of the TLB. In order to overcome such execution history dependency, current solutions provide the desired predictability by applying *register-based* MMU. This is the case of the PROARTIS industrial partner AFS, in which the case study provided in WP4 uses specific PowerPC registers called *Block Address Translation* (BAT) registers that can map linear chunks of memory as large as 256 MB. By doing so, the OS can map large portions of the address space ensuring that all the required page translations will be contained into the BATs. Therefore, the PROARTIS platform will contain a set of BAT registers in order to provide the required MMU support to execute the case study applications provided by the WP4.

However, because of the recent interest of Airbus in recent PowerPC products like the *e500 core* present in the MPC85xx and the new eight-core, the P4080, the PROARTIS platform will also focus on other MMU designs. These processors do not implement BATs but *book-e* MMU: a software-controlled refilled MMU implementing two TLBs, one classic with fixed-size 4 KB pages and another one with variable-sized pages, resulting more versatile than BATs although with less predictability. Thus, the PROARTIS platform will be extended so as to implement a *TLB-based* MMU.

3 Technical Specification: Software Technology (D2.1)

Deliverable 2.1 *Technical Specification: Software Technology* specifies the software technology (i.e. programming languages, Operating System, middleware) to use in the PROARTIS experimental developments, as well as consolidating the research requirements that justify this specification. In performing this work WP2 shall also meet the requirements emanating from WP4 for the development of the avionics case study and the requirements arising from WP1 and WP3 as applicable.
It is worth clarifying that, whereas WP1 and WP2 both focus part of their research at the software level, a clear boundary exists between their respective responsibilities: the run-time libraries addressed by WP1 exclusively apply to sequential (single-threaded) execution; conversely, WP2 addresses the run-time libraries that support concurrent (multi-threaded) execution under the execution models of specific interest to PROARTIS.

This deliverable summarizes the work done to date in Task 2.1 *Definition of baseline software technology.* All the objectives defined in this task have been achieved.

3.1 Specification of Research Objectives as Understood at Milestone 1

From the perspective of WP2, the research objectives of PROARTIS can be described as follows:

- **O2.1**: design, prototype and evaluate run-time software architecture solutions (i.e., components, data structures and algorithms) that – operating in conjunction with informed compilation – reduce and possibly annihilate the dependency of application timing on execution history. This objective will be pursued seeking time composability at the level of system components and time compositionality at the level of the system, in a manner that will prove economically viable for industrial application. This objective directly captures requirements **R53.4**, **R53.5**. **R53.7**, **R53.9** and **R53.10**.
- **O2.2**: design, prototype and evaluate run-time mechanisms and features that – operating in conjunction with informed compilation and thus in full alignment with the proceedings of WP1 – ensure that the execution behaviour of the application fits the assumptions made by analysis techniques developed in WP3.

Some explanatory observations on the above descriptions are in order:

i. The strategy adopted in WP2 for the evaluation of the results obtained against both objectives will be incremental and progress from laboratory experiments to industrial case studies.

ii. The term “run-time” as used in the context of WP2 denotes the whole set of software components comprised in the kernel and middleware layers underneath the application. This notion is important in two ways to the correct interpretation of objectives **O2.1-2**:

a. It sets a clear boundary between the responsibilities of WP1 and those of WP2, such that the former addresses the randomization solutions that are to be applied to sequential, single-threaded, software, typically via the compilation process, while the latter addresses those that need to be applied to concurrent software, i.e., in the presence of multiple threads of execution; for obvious reasons of economy, WP2 shall concentrate on a limited range of concurrency models, notably those that directly meet the requirements of WP4 and those additional ones, if any, that qualify for the development of critical real-time embedded systems.

b. It underlines an important character of the envisaged PROARTIS solution: that, following the principle of separation of concerns [Di82], the management and actuation of the software randomization solutions proposed by PROARTIS are completely hidden from the algorithmic structure of the application; hence, while the application designer may be offered specific configuration options to control or influence the way in which software randomization actions will take place, no direct control API will be provided to the application software to directly execute any such actions.

3.1.1 Rationale

Objective **O2.1** acknowledges one of the key challenges arisen from the requirements capture phase conducted in the first 6-month period of the project: to identify and defeat the sources of dependency of the WCET bounds on the execution history. This challenge reflects requirement **R11.1**.

In keeping with the state of the art, the definition of WCET is an attribute of the sequential execution of application level code between any two points of interest in any job of the tasks that compose the system. Such execution by definition includes all run-time system services executed on behalf of the application to service explicit synchronous calls of the application code of interest. It is precisely on account of this observation that WP1 and WP2 are complementary in their effort to incorporate software randomization solutions to such execution: WP1 will use compilation techniques and run-time libraries exclusively applied to sequential (single-threaded) execution, concretely those related to memory management, e.g. heap memory allocator, to inject software randomization in the application software, whereas WP2 will create specific variants of the concurrency-related run-time libraries that will support the concurrent execution of the application on the PROARTIS platform (see Deliverable D1.1). In the respect of Objective **O2.1** we can therefore draw this boundary: WP1 will address all aspects of sequential execution, as seen from the hardware and the software, hence via hardware design and modifications to the associated cross compilation system; WP2 will instead build on the sequential execution platform provided for by WP1 and will address all aspects needed to permit concurrent execution on it in a manner that preserves the PROARTIS properties of interest, that is to say, the elimination of dependency of timing behaviour from the history of execution.

Objective **O2.2** addresses the need to maintain full and complete alignment between the assumptions that inform the analysis theory chosen to validate the timing feasibility of PROARTIS applications and the behaviour that the application will actually exhibit at run time in the dimensions of interest to the said analysis. What these assumptions are in practice is still not fully defined at the time of this writing, but will become clear over time, expectedly in the early phase of the m7:m12 period of the project. What is absolutely apparent, however, is that *all* analysis models necessarily make some assumptions in the way they capture the aspects of reality that they aim to represent. It has become increasingly manifest to the developers of CRTES that the results produced by those analysis methods only hold as long as the assumptions that they are based upon also hold. Unfortunately, not all such assumptions can be assured by static configuration of the system: we shall make a few explanatory examples of this case in the paragraph below. If any single assumption of those that cannot be guaranteed statically is violated at run time, the results of the analysis, which were used as verification or even certification evidence, become invalid and the system execution devoid of the relevant guarantees. This is obviously not acceptable for critical systems.

Examples of assumptions that concern the analysis of the execution time behaviour of the application can easily be found at both the level of timing analysis (where the WCET bounds are determined) and the level of schedulability analysis (which ascertains whether the application deadlines can be met at execution time).

- Timing analysis assumes, for any given processor, a given hardware configuration, a given software configuration and a given system configuration. It is obvious that, in case any elements of the actual configuration set at run time for the given processor differ from those assumed, the WCET bounds previously obtained would have no value. It is interesting to notice in this regard that a processor whose timing behaviour exhibits dependency on the history of execution may dramatically amplify the need for assumptions and, consequently, the fragility of the analysis results. The whole of the PROARTIS innovation aims to defeat (or at least bound) this dependency.
- Schedulability analysis assumes, for example, that the WCET bounds can never be exceeded and also, depending on the accuracy targets of the analysis techniques in use, it also assumes what style of interaction (preemption, access to shared resources, etc.) can occur between application components at run time in order that the corresponding overheads can be correctly accounted for. It is obvious that if the actuality deflects in any manner from the assumptions on which the analysis is based, no guarantee holds any more on the ability of the application to meet deadlines at run time.

Objective **O2.2** of WP2 strives to eliminate the above risks to achieve full compliance with the assumptions retained by the timing and schedulability analysis methods endorsed by PROARTIS. To this end, WP2 will use a combination of preventive techniques, which achieve constructive assurance from the way the application is built, and run-time monitoring techniques, which offer the dynamic complement of the sought assurance.

3.1.2 WP2 Top-level Requirements

Based on the earlier discussion we can identify the following requirements to be satisfied by the project work packages so as to ensure a successful development of the PROARTIS research from the perspective of WP2:

- (**R21.1**) WP3 shall strive to develop their analysis techniques in a manner that singles out the fundamental assumptions on which those techniques rest.

- (**R21.2**) WP2 shall maintain full and complete alignment between the assumptions that inform the analysis theory chosen to validate the timing feasibility of PROARTIS applications and the behaviour that the application will actually exhibit at run time in the dimensions of interest to the said analysis.
- (**R21.3**) WP2 shall identify the main sources of dependency of the timing behaviour of system component to execution history in the internal structure of the OS and middleware layers that will form part of the PROARTIS infrastructure, in the form of run-time libraries incorporated in the executable by the compilation process described in deliverable D3.2.
- (**R21.4**) WP1 shall support the integration of the products of WP2 in the PROARTIS tool chain and help ensure their correct and efficient operation.
- (**R21.5**) WP2 shall address its objectives following an incremental approach, progressing from the build up of laboratory experiments to supporting the development of the avionics demonstrators carried out in WP4.
- (**R21.6**) WP3 shall devise ways to determine if the independency from sensitivity to execution history achieved by the techniques developed in WP2, in addition to those provided by WP1, makes the analysis results sufficiently trustworthy to meet requirement **R53.2**.
- (**R21.7**) The avionics demonstrators to be developed in WP4 shall be capable of determining whether the software design solutions proposed in WP2 are practicable, i.e., economically implementable and performance-wise viable in the context of certified avionics

3.2 Specification of Baseline Software Technology

One of the key challenges of PROARTIS is that we shall demonstrate its industrial value via some representative case studies. It is interesting to elaborate on the notion of representativeness. It is apparent that possibly massive changes to the hardware platform are necessary to inject the required level of randomization in the mechanisms of low-level execution: such changes, however, are likely to meet with industrial acceptance since CRTES industry is used to and prepared to cope with the discontinuity of hardware evolution. For software, instead, the less intrusive and extensive the needed changes are, the higher their likelihood of acceptance in industrial practice. These considerations suggest that the PROARTIS platform has ample latitude at the level of the hardware as well of the compiler and middleware, as long as little or no modifications are required to the application software.

C and Ada are the baseline language technology for PROARTIS. Both have industrial relevance and bid on the most practical route for exploitation. Two further reasons support the choice:

- the PROARTIS team has vast and long-standing experience in both languages and can thus easily and effectively handle legacy code offered by members of the IAB, in addition of course to partner AFS, to conduct representative experiments;
- the language technologies that dominate the respective markets are both based on the gcc back-end, natively for C, and by strategic decision in the AdaCore implementation of the GNAT Ada compiler [AC10]. This important technological intersection will ease the construction of the PROARTIS tool-chain and permit to

unify a good deal of it, with obvious benefits on the reduction and the concentration of effort and resources.

It is interesting to notice that the commonality highlighted above holds in fact for all target platforms supported by GNAT. Luckily – but not surprisingly given the penetration of Ada in the aerospace industry – the product offering of AdaCore extends to the PowerPC family. This is very convenient, since the AFS partner has required the PROARTIS case study to target the low-power, 32-bit implementation of the PowerPC Reduced Instruction Set Computer (RISC) architecture [PowerISA] specified in [MPC750].

3.3 Specification of the Operating System Kernel and Middleware Technologies

WP2 shall take an incremental approach to the investigation, design, implementation and experimental evaluation of the run-time system (understood in the extensive sense noted earlier in this section) components needed to accomplish objectives **O2.1-2**.

Work shall begin by looking at strictly sequential execution, which has minimal needs for run-time support and thus leaves ample room for introducing and evaluating software randomization features without risk of side or emerging effects. It is interesting to notice in this respect that one of the product options of the GNAT Pro for PowerPC offering by AdaCore is the so-called “zero-footprint” runtime, which is precisely intended for the development of strictly sequential Ada applications (those that are to feed individual partitions in compliance with the IMA standard) and it is thus stripped of all elements that serve the (otherwise much richer) execution semantics of the full Ada language. That particular version of the GNAT Pro product will be initially selected for the Ada baseline of the WP2 effort.

The subsequent step of work in WP2 will address restricted forms of concurrent execution in single-core processor architectures.

- The Ada baseline instead will use the Ada runtime restricted to the Ravenscar Profile [BDV03], which implements a concurrent computational model especially fit for CRTES, directly amenable to static analysis for both timing and schedulability, and placing very modest requirements on the run-time support. The fairly small size (in the region of a few thousands lines of code) of the needed run-time system and the familiarity with it owned by the UNIPD partner, leader of WP2 will permit to operate directly on the kernel sources and modify them where appropriate, without thus resorting to stubbing as will be done for the C baseline. Interestingly, the UNIPD team leader was among the promoter, early reviewer and first user of the port of Ada Ravenscar run-time to the ERC32 and LEON processors selected by the European Space Agency as their technology baseline for space applications. The fact that the AdaCore's product offering adheres to the free and open source software (F/OSS) model further eases the access of the PROARTIS team to the sources of the entire Ada technology baseline, from the compiler to the run-time libraries. The product of this line of action in WP2 will be used and verified in WP2, using additional UNIPD resources if needed and with support and collaboration from AdaCore, member of the AIB and supplier of the baseline Ada technology.

The third and final step of the work in WP2 will try to investigate how and to what extent the solutions investigated in the first two steps scale to a multi-core processor architecture. More reflection on how to formulate and attach this challenge is however needed.

The potential sources of dependency on execution history that may exist in the run-time libraries needed to support both the C and the Ada baselines set out above will be identified and documented in the m7:m12 period of the project. This may appear somewhat out of sync with the progress of WP1. In fact, this follows from the fact that the essential hardware elements of interest to PROARTIS are so obviously identifiable to the point of permitting early analysis (as discussed in D1.1). In contrast, the extent of software structures and solutions that can be used for the OS and middleware levels of CRTES is so vast and diverse that no useful analysis can be afforded before baseline selections have been made. For this reason therefore the WP2 effort in the m1:m6 period has been spent in the direction of capturing the strategic premises, both scientific and industrial, of the PROARTIS research and investigating the technical needs arising from the other WP.

3.4 Specific Avionics Demonstrators Requirements

The approach envisioned for the development of the PROARTIS avionics demonstrators is discussed in WP4. In this section we recapitulate the most critical requirements which that approach places on WP2.

- The proceedings of WP2 shall feed the three incremental steps to be taken in WP4, i.e., (i) sequential execution only; (ii) ARINC-compliant single-partition threaded execution; (iii) ARINC-compliant multi-partition threaded execution. Extension to multicore platforms will be attempted in a best effort manner contingent on the residual energy of WP2 at that time.
- In order to achieve this goal, WP4 shall enumerate: (i) the A653 threading capabilities to be supported by WP2; (ii) the I/O services that may involve blocking and that will be invoked by the application software of the avionics demonstrators; (iii) the A653 intra-partition services needed to support the third increment of the avionics demonstrators.

4 Technical Specification: Probabilistic Timing Method Selection (D3.1)

Deliverable 3.1 *Technical Specification: Probabilistic Timing Method Selection* specifies the requirements of the timing methods to be used in the project. It also provides the research requirements that justify this specification. This deliverable summarizes the work done to date in Task 3.1. *Selection of probabilistic timing methods*. All the objectives defined in this task have been achieved.

4.1 Specification of Research Objectives as Understood at Milestone 1

From the perspective of WP3, the research objectives of PROARTIS can be described as follows:

- **O3.1**: design, implement and validate probabilistic timing analyses that will provide estimations of WCET as well as schedulability tests;
- **O3.2**: design, implement and validate statistical analyses that will provide estimation of WCET, proofs of good randomness, as well as schedulability tests.

We give below some explanatory observations on the terms we use to define the objectives **O3.1** and **O3.2**; these observations complement those provided in Section *1.2. Taxonomy of Analysis Methods.*

We will make use of the words "*probabilistic analysis*" or "*statistical analysis*" depending on the approach on which the solution is based. Both terms relate to the study of the relative frequency of events, however there are fundamental differences between them [Ski01]:

- A *probabilistic analysis* is based on the probability theory and it is an analysis that provides the probability or chance of occurrence of *future* events. For instance, on the game of throwing a dice, assuming that the dice is not loaded, a probabilistic analysis of the "dice model" would assume that the probability of each number appearing is 1/6.
- *A statistical analysis* is based on the statistics theory and it is an analysis that searches for a model or some properties when studying some (often large) mass of data of *observed past* events. For instance, on the same example as above, in order for the statistician to check if the dice is loaded, he/she would define the hypothesis that "the probability of each number is the same". By analyzing a large number of throws, this latter hypothesis will be accepted or rejected with a level of confidence.

In order to avoid confusion, in this document we will not use the word "stochastic" that is often associated with unpredicted behaviour (as opposed to a deterministic). Even though the term is usually described to mean "based on the theory of probability" the term has been abused and therefore it will be avoided.

The first papers in the real-time community related to our work had equally used the words "stochastic analysis" [GaLu99], "probabilistic analysis" [TDS+95], "statistical analysis" [AtBe98] and "real-time queuing theory" [Leho90]. Since the paper of Diaz et al. [DGK+02], the "stochastic analysis" of real-time systems has been used regularly by the community regardless of the approach (probabilistic or statistical). We note that the terminology used in the past is somehow inconsistent; one of the objectives of the project is to define precisely those terms so that they can be widely used.

4.2 WP3 Top-level Requirements

Based on the objectives analysis exposed above we identify requirements to be verified by WP3 and the other work packages:

- (**R31.1**) WP3, in collaboration with WP1, has to ensure that the timing behaviour of the proposed hardware and hardware-dependent techniques fits the hypotheses of the analysis provided once **O3.1** and **O3.2** are fulfilled. Such requirement will be covered by objective **O1.3** defined in deliverable D1.1.
- (**R31.2**) WP3, in collaboration with WP2, has to ensure that the execution behaviour of the application fits the hypotheses of the analysis provided once **O3.1** and **O3.2** are fulfilled. Such requirement will be covered by objective **O2.2** defined in deliverable D2.1.
- (**R31.3**) WP3, in collaboration with WP4, has to ensure that the assumptions and analyses done in WP3 are applicable to the case of certified avionics.
- (**R31.4**) WP3 has to provide a tutorial explaining the terminology and proposing a taxonomy of different techniques, being them probabilistic or statistical. The tutorial will also explain the application of these techniques to PROARTIS and the assumptions they made. Such requirement will be covered by objectives **O3.1**, and **O3.2** defined in deliverable D3.1.

4.3 PROARTIS timing analysis of critical real-time systems on randomized HW and SW platforms

During the first six months of the project, we have identified three different flavours of methods to analyse the WCET of systems with random timing behaviour:

- MBPTA: In *Measurement Based Probabilistic Timing Analysis* complete runs of the software components under study are done. From these runs we collect data about the timing behaviour of the software component under study when run on the randomizing low-level software and hardware elements of the PROARTIS platform. This information is used to determine probabilities of individual elements of the system, and then use these as inputs to the timing analysis of the overall system. We will apply the term Measurement Based to all analysis that derive a-priori probabilities from a measurement based model.
- SPTA: In the *Static Probabilistic Timing Analysis,* probabilities of individual operations, or components are determined “statically” from a model of the processor or software. The design principles of PROARTIS will ensure that it makes sense to derive such probabilities and make assumptions of independence. For example, one of the earlier analysis done in the project relies on defining a cache architecture for which it is possible to compute *a priori* the probability of hit or miss based simple on a distance metric. This enables the calculation of the distribution of the number of cache hits and misses for a straight piece of code. And from this information and values of the time penalties for cache hits and misses, an static calculation of the distribution of execution times can be performed. We will apply the term Static to all analysis that derive the a-priori

probabilities from an abstraction of the system.

- StA: In Statistical timing Analysis, hypothesis about the behaviour of the system are formulated and tested using a set of statistical techniques. In particular, hypothesis to test are the level of dependency (e.g. test of independence, tests of correlation, other levels of dependency), quality of random number generators, sensitiveness of the approaches to quality of the random number generators, etc. Note that the main emphasis of the project is on the behaviour of the tails (extremes) and this determines the types of statistical analysis that can be applied.

These steps (in light grey) as well as their relations with the tools (in black) presented in D3.2 are illustrated in Figure 2.

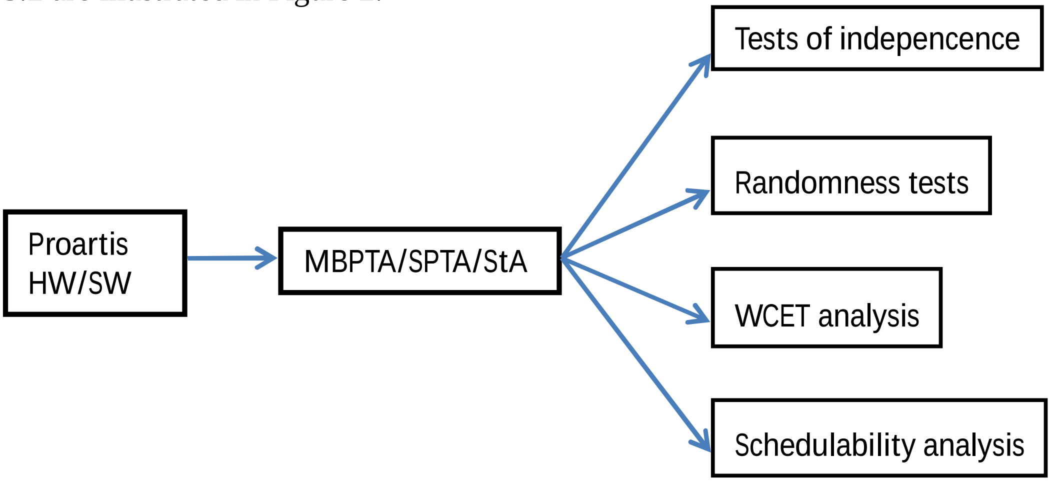


Figure 2. Relationships of the timing analysis within PROARTIS

4.3.1 Quality tests for the random generator and independence tests

These tests concern the quality of the random generator and the independence of the data provided by the measurement based-probabilistic WCET analysis and the statistical probabilistic WCET analysis.

The quality of the “random generator” is crucial since it might impact the rest of the analysis, e.g., if there is some unknown relation between data provided by the generator to the simulator then a statistical analysis applied at the level of the WCET estimation might find properties that are false. By the “random generator” we mean the HW and SW techniques that introduce randomization in the system. The simplest one is an on-chip random number generator, other components can be hashing tables of memory addresses. The project will develop specific experiments and analysis of the sensitivity of the quality of the random number generators. It is hypothesized that the impact is minimal, however, sufficient evidence needs to be provided to support the certification arguments.

Independence tests are needed because most of the theoretical results that we will use make the assumption that the random variables are independent. The independence tests can also be used to test the goodness of fit or to define the quality of a random generator. We will use the *chi-square test*, which is the most widely known independence test. One limitation of this test concerns the correlation of data, i.e., the tested data must be obtained from independent observations/experiments. Moreover the chi-square test might fail if a value that the variable might take has a small number of occurrences in the sample. The input for these tests will be the data provided by the random generator, the MBPTA and the SPTA. The output will be either the confirmation of independence or the relations that might exist among data. In the latter case, the relations should be defined such that they make it possible to modify the architecture randomization in order to provide new (independent) data.

4.3.2 Statistical timing analysis (of critical real-time systems)

PROARTIS analysis is based on the fact that the statistical theory for real-time systems must take into account the worst-case values (be them execution times or response times) as rare events. Therefore in PROARTIS we will use statistical theory providing information on the tail events (highly unusual). Such results belong to the theory of rare events and we are interested in extreme values theory [BTG+04] and large deviation theory [DeZe97].
Concerning the extreme values theory, we are interested in using the Gumble distribution among the three extreme value distributions (Gumble, Frechet and Weibull) [Ol02]. A first step toward fitting the data to this particular distribution is needed and errors could be introduced during this step (thus different levels of confidence might be proposed). One solution to decrease the errors is proposed in [HaHM09] and it is based on block maxima (grouping different values within a distribution into the maximal one).

We will also use the large deviation theory. This theory deals with tail events too and extends the Central Limit Theory. Thus it has the feature that it can be only applied to a sum of *independent and identically distributed* (IID) random variables. This property could be useful for instance when we have traces for different parts of a program and we would like to combine them.

We underline here that the both approaches make the assumption that the random variables are IID. Even if intuitively these approaches are more suited to the estimation of the (worst-case) execution times, they could be also used to study the schedulability of real-time systems. Nevertheless such results are rare and to our best knowledge there are only two papers presenting them [NaCS07] and [Binn04]. A paper that is at the crossroad of the two fields (schedulability analysis and worst-case execution time estimation) is [KhHl10] that uses re-sampling techniques for WCET to ease the schedulability analysis.

The first paper considering the statistical estimation of worst-case execution times is [EdBu01] in which extreme value statistics are used to model the behavior of caches, more recently [HaHM09] improved the analysis, addressed flaws in the previous work and produced a refined method also based on extreme value statistics.

All papers presented before consider only sequence of independent and identical distributed random variables. This hypothesis is not always realistic and in order to take into account dependencies the authors propose in [BeBN03] an approach using copulas.

4.3.3 Probabilistic analysis (of critical real-time systems)

We will use the probabilistic analysis especially to decide the worst-case execution time and the schedulability of the system, i.e., to answer the question “are the deadlines met and with what probability when the WCET are random variables?”. Therefore this analysis will have as input WCET estimations (provided as random variables) for each program and the output will be the probability of missing the deadline for the set of programs.

The first results on probabilistic schedulability analysis were proposed in [Leho90] and the author extends the queuing theory under real-time hypotheses. Even if this work was furthermore improved by results such [ZHL+02], the main limitation concerns the use of the same probability law for the execution times of all programs. This latter hypothesis is not always realistic, e.g., the traces of execution of two different programs could fit (after a WCET estimation) to different probability laws.

Another relevant work [AbBu99] proposes an interesting definition of probabilistic deadline, but the authors consider a special scheduling model providing isolation between tasks. The results presented in [GaLu99], [TDS+95] and [AtBe98] consider also execution times as random variables and special assumptions are made on the critical instant.

The main result of this area was proposed in [DGK+02], but the analysis is difficult to use in practice. An improvement based on re-sampling was proposed in [KhHl10] and this work belongs more to the statistical analysis of real-time systems. Within PROARTIS we would like to propose improvements of these two results, e.g., choice of a re-sampling technique that uses properties of the obtained WCET estimations.

Except for some known probabilistic results (like stability of Markov chains) the results used for the schedulability of real-time systems are mainly based on operations with random variables that extend deterministic calculation. One exception is the queuing theory, but it is usually classified as operations research.

5 Baseline Integrated Tool-chain (D3.2)

Deliverable D3.2 “Baseline Integrated Tool-chain” describes the complete envisioned tool-chain infrastructure upon which the PROARTIS research will be conducted. The PROARTIS tool-chain comprises the following components: architectural simulator (including both functional and timing simulator), compiler, real-time operating system (RTOS), runtime libraries and the timing analysis tool (see Figure 4).

Three main criteria have been used to select the components of the tool-chain:

- *Maturity*. The following questions have been considered: How mature is the component? How many users are using the component? By doing so, our confidence in the quality (i.e., accuracy, performance, stability, usability) of the tool increases.
- *Functionality*. The following question has been considered: How easily (and practically) can we implement the new capabilities/functionalities required to develop the PROARTIS research?
- *Overhead of understanding/changing the component*. Too feature-rich or complex tools may slow our development down and cause delays and obstructions in our effort to achieve the PROARTIS goals.

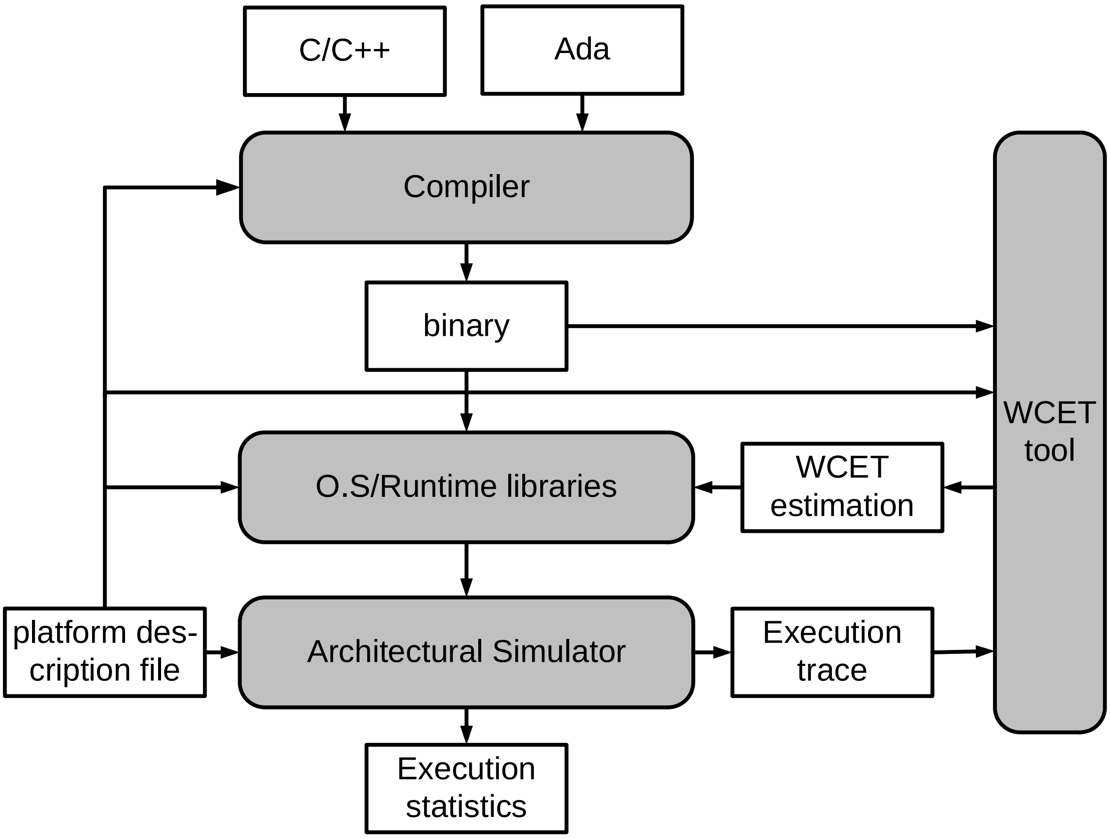


Figure 4. Block Diagram of the PROARTIS tool-chain

5.1 Hardware simulation tool (architectural simulator)

Several alternatives have been considered for the selection of the baseline architecture simulator. In particular, we looked at:

- ReSP [RESP]. This simulation framework fulfils the project requirements in terms of functionality and overhead; however, talking with the developers we learned that it still was at a very early stage of development. For this reason this tool was discarded.
- MERASA [MERASA] Simulator. This simulator was immediately discarded because its implementation is highly dependent to the Tricore instruction set architecture, and it does not have (and makes it difficult to include) the functionality stated in the WP4 requirements.
- SoCLib [SOCLIB]. This simulation framework is an open source tool used by several European institutions. Its design eases changeability and it also supports the PowerPC ISA, which is the baseline selected in the project. This fulfils the three main criteria defined above (maturity, functionality, overhead) and so *SoCLib has been included in the PROARTIS baseline.*

The 32-bit PowerPC instruction set with classic “AIM” FPU support as the UISA assume by the PROARTIS avionics demonstrator.
To that end, the reference processor considered in the architectural simulator is the MPC750 processor [MPC750]. It is worth noting that we do not aim to develop a cycle-true accurate simulator of the MPC750, but to have a functional interface equivalent to the MPC750 (Figure 5 shows the register interface required for the correct execution of the avionics case study). Unfortunately, SoCLib does not fully support the MPC750 functionality, but having the PPC405 core model, for which register level and functional models are available. This baseline processor model will then be incrementally improved to develop a MPC750-like core (FPU/MMU/instructions set).

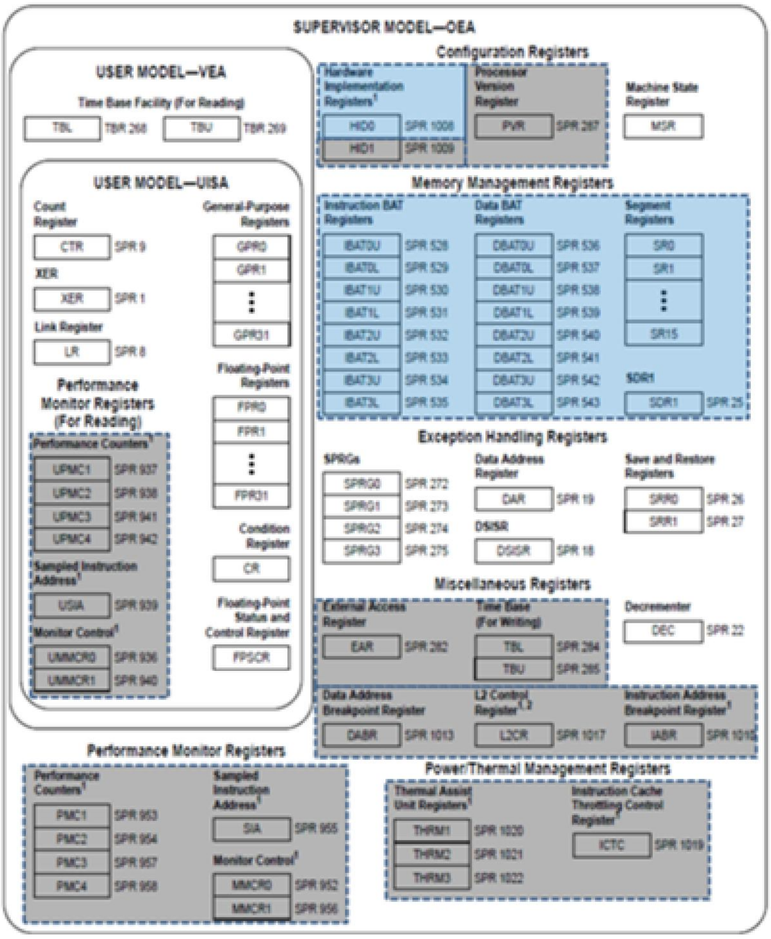


Figure 5. *MPC750 processor programming model (register sets). In grey, the registers not required to be supported by the simulator. In blue, the registers that may be required to be supported.*

5.2 Compiler

Two main constraints are placed on the compiler infrastructure at the front end and back end level:

i. On the front end, it must support the programming languages that we plan to use in the project: C and Ada (see Section 3.2 in Deliverable D2.1). The investigations made in the m1:m6 phase of the project have confirmed the language selection already anticipated in the Description of Work. At the time the proposal was being written, the programming language candidates were C and Ada. The rationale of their selection was the dominance and relevance that those languages have earned in the critical real-time embedded systems industry, although from very distinct origins and paths. Looking at the industrial domains represented in the Industrial Advisory Board of the project, C dominates in automotive and has an increasing presence in aerospace and ground transportation, whereas Ada is strong in aerospace and defense and significant in ground transportation. The wisdom of this selection has been strengthened by the observation that the language technologies earmarked for use in the project both base on gcc, which greatly facilitates the work of PROARTIS.

ii. On the *back end*, the selected compiler technology targets the PowerPC architecture.

On the basis of these considerations, PROARTIS embraces C and Ada as the baseline programming languages and use compiler incarnations for them that target the MPC750 using the same gcc compiler infrastructure. In particular:

- C baseline: gcc version 4.4 or 4.5**.**
- Ada baseline: GNAT Pro for PowerPC at versions 6.3.1-2, both of which use gcc 4.3.5. It is anticipated that the 2011 release of the GNAT Pro for PowePC will instead use gcc 4.5.1. The opportunity to switch to the technology base of the 2011 version will be considered in the course of the project. To increase the industrial impact of the PROARTIS project, we liaised with AdaCore (member of the IAB) to obtain a free license of their commercial compiler and runtime technology for PowerPC targets.

A further important requirement is that the selected compiler technology permits to introduce randomization as described in D1.1. To that end, LLVM [LLVM] has been selected to facilitate the PROARTIS research at the compiler level.

LLVM is a gcc back end compiler, which includes a collection of modular, reusable, extendable and adaptable libraries that operate on the intermediate representation (IR) produced by the gcc front end compiler, and can thus produce “regular” executables employing the randomization algorithms developed in WP1.

This solution works perfectly for PROARTIS because both the C and the Ada front end compilers use the same gcc IR format, which can therefore be taken the PROARTIS adaptation of LLVM. Hence, the randomization effects needed to pursue the PROARTIS goals will be introduced by the LLVM pass of the compilation process. The compiler infrastructure baselined in PROARTIS is shown in Figure 6.

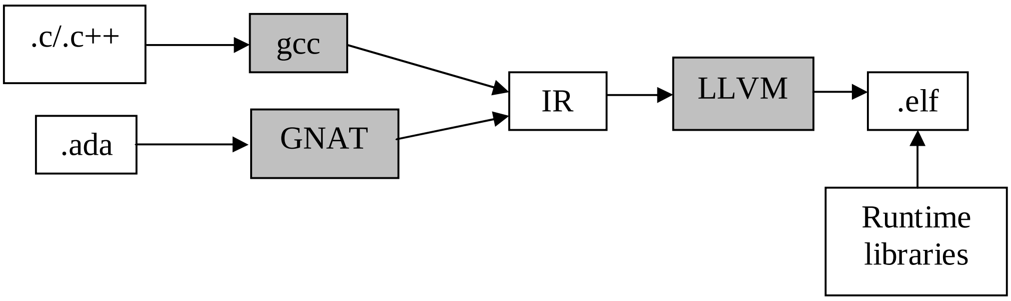


Figure 6: Block diagram of the compiler infrastructure

Further modifications, as outlined in D2.1, will be made to the run-time libraries, to make sure that the randomization effects are preserved in the face of concurrency.

5.3 Operating System

The timing behaviour of an individual task does not depend solely on the processor architecture, but also on the interfering presence (in space) and execution (in time) of multiple layers of software, including the OS and its asynchronous I/O services.

To address the issues that may get in the way in the pursuit of the PROARTIS goals from classical CRTE multi-tier software architectures, we shall follow an incremental approach. We will start from a purely sequential execution (and thus with smallest possible needs in terms of OS) so as to operate in a context in which we know and can control all the sources of interference on the timing measurements that we need to make. This initial OS layer will be represented as a thin module added on top of SoCLib. Subsequently, we will introduce more and more OS services, in a manner that permits us to bound the potential interferences that they may introduce.

Specifically, we will initially focus on non-concurrent applications so most of the OS services are not required. As we will move to concurrent applications and multicore architectures all required OS services will be incrementally introduced. To address restricted forms of concurrency, we shall proceed as follows:

- For the C baseline, the reference concurrency model shall adhere to the IMA ARINC 653 standard, which is the domain-specific standard embraced by the aerospace industry.
- For the Ada baseline instead will use the Ada runtime restricted to the Ravenscar Profile [BDV03], which places very modest requirements on the run-time support. The fairly small size (in the region of a few thousands lines of code) of the needed run-time system and the familiarity with it owned by the UNIPD partner, leader of WP2 will permit to operate directly on the kernel sources and modify them where appropriate, without thus resorting to stubbing as will be done for the C baseline.

5.4 WCET tools

The set of tools required to do the exploration mainly consist of:

- A set of scripts to parse the timing results obtained with the architectural simulator. In particular, program traces must be generated providing information regarding the type of instruction, its potential latencies and some other data required to perform static probabilistic analysis. For instance, static probabilistic analysis of the latency of data memory accesses may require a trace with all load and store instructions including the address being accessed and the reuse distance (distance between two consecutive accesses to the same cache line).
- A set of libraries (provided by RAPITA) for managing floating numbers with arbitrary precision. Actual precision is limited by the standards implemented by the instruction set architecture of the processor where the analysis tools are run. However, such precision is not enough for our purposes, and therefore, libraries implementing arbitrary precision are required. Previous experiments [BeBN03] have shown that the resolution of standard floating point arithmetic (even 64 bit floating point numbers) is not adequate. This is generally due to the fact that the repeated application of multiplications of floating numbers necessary to perform the convolutions accumulates floating point errors which can be larger than the range of probabilities that we are aiming at calculating.
- A set of libraries (provided by RAPITA) for managing Execution Time Profiles (ETPS). The amount of data to be analysed grows exponentially with the input traces. Some libraries managing ETPS bound the amount of data used by reducing the accuracy of those sets of data where high precision is not required. Thus, high precision data is kept only for those execution times really critical for the purposes of PROARTIS. In addition tools for operating with profiles very efficiently need

to be provided. Experiments based on standard convolution methods can take hours to compute even for simple programs. Alternative fast convolutions can be defined that exploit properties of the profiles to implement these convolutions in at least one order of magnitude reduction on computation time.

- Statistical Analysis tools, mainly toolboxes from MATLAB [MATLAB] and SPSS [SPSS]. Some purely statistical tasks such as tests of independence and fitting data into well-known functions are performed by means of state-of-the-art tools with statistic capabilities. Several additional software will be used, mainly for statistical analysis like R [R], WINBUGS [WIN], SAS [SAS], SPAD [SPAD], Maple [MAPLE]) .

5.5 Tool-chain installation

In order to ease the installation of the toolchain across all the partners, we have created a fully-automated script that carries the installation out, by the use of a configuration, building and installation tool named *`waf`*. Installation package provides the following:

- the SoCLib source file tree and its building tools: this is a library that provides hardware components of our architecture simulation tool like the processor, the Network-On-Chip, etc
- the source files of our PROARTIS_sim simulator : patches for the SoCLib source files with the modifications needed by our needs and new hardware components
- the source files of the benchmarks used for the first iteration: Mälardalen benchmark suite (http://www.mrtc.mdh.se/projects/wcet/benchmarks.html)
- the binaries of LLVM cross compiler and binutils for PowerPC
- a compression library for the trace files generated by our simulator provided by Rapita Systems.

5.5.1 Software Dependencies

The following software elements are required for a successful installation of the PROARTIS infrastructure.

- *`SystemC`* development files and 32-bit library (even in 64-bit machines, since the generated simulator must be linked with a proprietary 32-bit compression library for the generated traces)
- *`SDL`* development files and 32-bit library
- Shared *`GNU C LIBRARY`* >= 2.8 (Cross Compiler Dependency)
- *`Python`* >= 2.4
- a *`sh`* compatible shell, like *`bash`*

5.5.2 Installation

- Extract *Proartis_sim.x.tar.gz*:

```
> tar xvfz Proartis_sim.x.tar.gz
```

- **cd** to the extracted directory:

```
> cd Proartis_sim.x
```

- Configuration: type:

```
> ./waf configure ARGS
```

  To configure the project: If missing dependencies are found, resolve them by installing and configure the project again. The following arguments can be passed to the previous command:

  - **--with-systemc=SYSTEMCDIR**: specifies the *SystemC* installation directory
  - **--cross-compiler-path=CROSSDIR:** path in which the cross compiler will be installed
  - **--soclib-path=SOCLIBDIR**: path in which SoCLib will be installed
  - **--PROARTIS_sim-path=PROARTISSIMDIR**: path in which *Proartis_sim* files will be installed
  - For a complete list of the supported arguments type:

```
> ./waf –help
```

- Build: type:

```
> ./waf build
```

  to create the necessary configuration files, before installation

- Install: If privileged directories have been specified for installation during configuration, be sure that you have the right privileges before this command. Type:

```
> ./waf install
```

  to install the cross compiler and the simulator files.

- If you use *bash* type source *~/.bash_login* to load the environmental variables that have been added to it. If you use another shell, add the environmental variables of this file to the initialization files of your shell.

5.6 Tool-chain Use – First Iteration

As a part of the bootstrap phase of the project we integrated all the tools required to carry out all PROARTIS investigations. These tools include: our simulator PROARTIS_sim which uses SoCLib source files tree and its build tools, RapiTime and LLVM cross compiler and utilities for PowerPc.

After the integration phase, we carried out an iteration of the complete tool chain. In this first iteration, we used several Mälardalen benchmarks, which is a widely accepted benchmark suite for real-time platforms. Due to the lack of Floating point Unit in the current version of our simulation tool, we tested only the integer benchmarks.

The objective of this first iteration is to show how we can run a C application, instrument it by RapiTime, compile it with LLVM, run it on our simulation tool, from which we generate an execution trace that finally is fed to RapiTime. Finally, RapiTime generates a WCET estimation for the application. As an example, Figure 7 shows the WCET estimations obtained with RapiTime.

In the generated report from RapiTime we can inspect several information about the executed program from both static analysis of the code and the measurement information extracted from the generated trace. For example, we can obtain the WCET estimations of the overall program as well as the WCET estimations of specific program segments like functions. Moreover, WCET estimations of functions can also take into account the impact that the outer calls have on the overall execution function body (column *W-OverET* in the table shown in the Figure 7) or exclude the impact of outer calls (column *W-SelfET* in the table shown in the Figure 7). In our particular case we can see that the WCET estimation for the function Multiply which is the core of *matmult* benchmark is 749557 cycles. Finally, the graph shown in the snapshot shows, for each function of the benchmark, the *minimum* (yellow), *average* (green) and *maximum* (red) observed Execution Time, and the WCET estimation (blue). These results are just for illustrative purposes of the complete integration of the toolchain.

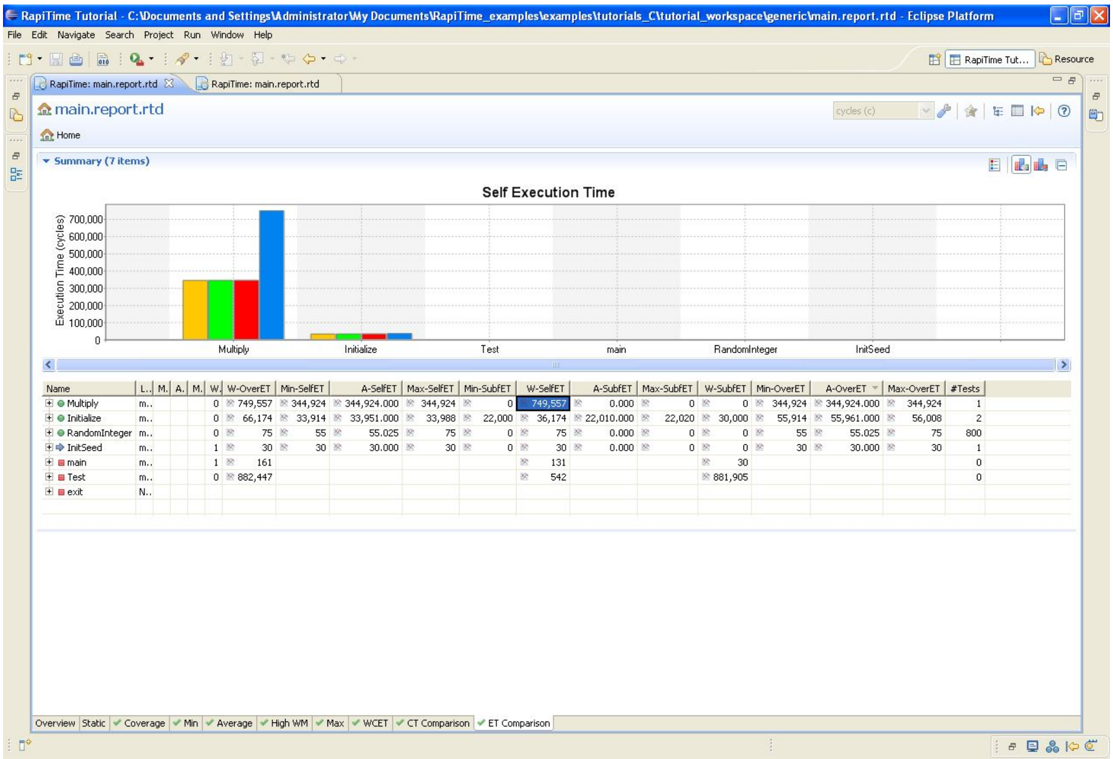
5.6.1 Requirements

- A Windows machine with Rapitime tool installed and `sshd` service running.
- *`Scp/ssh`* programs on the linux machine on which PROARTIS_sim is installed.

5.6.2 Setup

- Windows machine:

- Create an account to be used for instrumentation and report viewing or select an existing one.
- Install *`Rapitime`* program and *`Rapitime Examples`*
- Install *`OpenSSH`* and setup the *`sshd`*
- Configure your firewall to leave port 22 open



Name	L..	M.	A.	M.	W.	W-OverET	Min-SelfET	A-SelfET	Max-SelfET	Min-SubfET	W-SelfET	A-SubfET	Max-SubfET	W-SubfET	Min-OverET	A-OverET	Max-OverET	#Tests
Multiply	m..				0	749,557	344,924	344,924.000	344,924	0	749,557	0.000	0	0	344,924	344,924.000	344,924	1
Initialize	m..				0	66,174	33,914	33,951.000	33,988	22,000	36,174	22,010.000	22,020	30,000	55,914	55,961.000	56,008	2
RandomInteger	m..				0	75	55	55.025	75	0	75	0.000	0	0	55	55.025	75	800
InitSeed	m..				1	30	30	30.000	30	0	30	0.000	0	0	30	30.000	30	1
main	m..				1	161					131			30				0
Test	m..				0	882,447					542			881,905				0
exit	N..																	

Figure 7: Snapshot of the RapiTime tool

- Install the necessary files for the collaboration between *`PROARTIS_sim`* and *`Rapitime`*

An alternative way to obtain the same result is to use a virtual machine with everything installed, provided by BSC. In this case the only need is to configure the linux machine to redirect connections from a port to virtual machine's port 22.

- Linux machine:

- edit *`soclib_components/rapitime_makefile.mk`* by setting the following variables:
  - **`USERNAME`**: username of the account on the Windows machine which is used for instrumentation
  - **`HOST`**: name or ip of the Windows host
  - **`PORT`**: port used for by *`sshd`* service in Windows host
- You may want to configure *`ssh`* to remember the password of your Windows account in order to avoid retyping it every time a file is transferred or a remote command is invoked.

5.6.3 Usage

- Boot the Windows machine in which Rapitime is installed and login to the account which has the privileges to run the *sshd* service.
- In Linux machine:
  - **cd** to PROARTIS_sim directory.
  - Select the application which you want to instrument and compile and set it in the variable **allsubdirs** of the *soft/Makefile*. Specify only ONE application in this variable. For example, if we want to instrument and compile the matmult benchmark of the *Mälardalen* suite, which implements an integer matrix multiplication we set it as following:

    ```
    allsubdirs = matmult
    ```

  - Build the simulator by typing:

    ```
    make
    ```

  - Instrument the source files specified in the applications' Makefile and compile the instrumented application by typing:

    ```
    make instrument
    ```

  - Edit the *param_file*, configuring the simulator's parameters and be sure that

    you specify the applications's *bin.soft* file to the **-executable** option. For example:

    ```
    -executable=soft/matmult/bin.soft
    ```

  - Run the simulation by typing:

    ```
    ./PROARTIS_sim param_file
    ```

  The simulation will generate a compressed trace file with the name *test-trace.rpz*.

  - Move the trace file to the Windows machine and create a report by typing

    ```
    make generate_report
    ```

  - View the report in the Windows machine by opening *Rapitime* from ***Start menu→Programs→Rapitime Examples→Start Tutorial*** and typing in a terminal:
    - **go.bat**
    - **main.report.rtd**

References

[AbBu99] L. Abeni and G. Buttazzo. QoS Guarantee Using Probabilistic Deadlines. In Proceedings of the 11th Euromicro Conference on Real-Time Systems (ECRTS99), pp. 242-249, June 1999.

[AC10] AdaCore, Product offering for PowerPC embedded targets, http://www.adacore.com/home/products/gnatpro/development_solutions/safety-critical/do-178b/powerpc/

[AMBA] http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.set.amba/index.html AMBA bus Specifications (rev. 2.0, 3.0 and 4.0). ARM

[AtBe98] A.K. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In 19th IEEE Real-Time Systems Symposium, 1998.

[BDV03] A. Burns, B. Dobbing, and T. Vardanega. Guide for the Use of the Ada Ravenscar Profile in High Integrity Systems. TR YCS-2003-348, University of York. 2003. Also: ISO/IEC TR 24718:2005, http://standards.iso.org/ittf/PubliclyAvailableStandards/index.html

[BeBN03] Guillem Bernat Alan Burns and Martin Newby, Probabilistic Timing Analysis: an Approach using copulas, Journal of Embedded Computing, 2003

[BeCP02] G. Bernat, A. Colin, and S. M. Petters. WCET analysis of probabilistic hard real-time systems. In Proceedings of the 23rd Real-Time Systems Symposium RTSS 2002, pages 279- 288, Austin, 2002.

[Binn04] Pam Binns, Statistical Estimation of Aperiodic Response Times when Scheduled on top of Static Timelines, In Proceedings of the 1st International Workshop on Probabilistic Analysis Techniques for Real-Time and Embedded Systems, PARTES'04, Pisa, 2004

[BTG+04] Jan Beirlant, Jozef Teugels, Yuri Goegebeur, Johan Segers, Daniel De Waal and Chris Ferro, Statistics Of Extremes: Theory And Applications, Wiley, 2004

[DeZe97] Amir Dembo and Ofer Zeitouni, Large Deviations Techniques and Applications (Stochastic Modelling and Applied Probability), Springer, 1997

[DGK+02] J.L Diaz, D.F. Garcia, K. Kim, C.G. Lee, L.L. Bello, Lpez J.M., and O. Mirabella. Stochastic analysis of periodic real-time systems. In 23rd of the IEEE Real-Time Sys- tems Symposium, 2002.

[Di82] E.W. Dijkstra. On the role of scientific thought. In Dijkstra, E.W., ed.: Selected writings on Computing: A Personal Perspective. Springer-Verlag New York, Inc. (1982) 60-66 ISBN 0-387-90652-5.

[EdBu01] S. Edgar and A. Burns. Statistical analysis of WCET for scheduling. In Proceedings of the 22nd IEEE Real-Time Systems Symposium, 2001.

[FLEX] http://www.flexray.com/

[GaLu99] M.K. Gardner and J.W. Lui. Analyzing stochastic fixed-priority real-time systems. 5th International Conference on Tools and Algorithms for the Construction and Analusis of Systems, 1999.

[Gold91] David Goldberg, What Every Computer Scientist Should Know About Floating-Point Arithmetic, ACM Computing Surveys, 1991

[HaHM09] Jeffery Hansen, Scott Hissam and Gabriel A. Moreno, Statistical-based WCET estimation and validation, ECRTS 2009 9th International Workshop on Worst-Case Execution Time (WCET) Analysis

[Ja08] B. Jacob, S. W. Ng, and D. T. Wang. Memory Systems: Cache, DRAM, Disk. Morgan Kaufmann, 2008.

[JEDEC08] JEDEC Solid State Techn. Assoc. JEDEC DDR2 SDRAM Specification JEDEC Standard No. JESD79-2E, April 2008.

[KhHl10] Khaled Refaat and Pierre-Emmanuel Hladik. Efficient stochastic analysis of real-time systems, In 22nd Euromicro Conference on Real-Time Systems, (ECRTS2010), July 2010

[Leho90] J.P. Lehoczky. Real-time queueing theory. In 10th of the IEEE Real-

Time Systems Symposium, 1990.

[LLVM] The LLVM Compiler Infrastructure. http://llvm.org/

[LuSt99] Timing anomalies in dynamically scheduled microprocessors. Thomas Lundqvist and Per Stenstrom. In RTSS '99: Proceedings of the 20th IEEE Real-Time Systems Sym-posium, Washington, DC, USA, 1999.

[MAPLE] Math Software for Engineers, Educators and Students - www.maplesoft.com/

[MATLAB] http://www.mathworks.fr/

[MERASA] MERASA (Multi-Core Execution of Hard Real-Time Applications Supporting Analysability). EU-FP7 Project: www.merasa.org, 2007.

[MPC750] Freescale Semicondutor Inc. Advance Information MPC750 RISC microprocessor Technical Summary. 1997

[MPCFPE32] Programming Environments Manual for 32-Bit Implementations of the PowerPC™ Architecture MPCFPE32B, Rev. 3, 9/2005 (http://www.freescale.com/files/product/doc/MPCFPE32B.pdf)

[NaCS07] Navet N., Cucu L. and Schott R., Probabilistic estimation of response times through large deviations, 28th IEEE Real-Time Systems Symposium (RTSS'07), Work in Progress session, Tucson, December 2007

[Ol02] Foundations of Modern Probability, 2nd ed. Springer Series in Statistics, 2002

[PowerISA] PowerPC Instruction Set Architecture specification 2.06. Book I. http://www.power.org

[QBB+09] E. Quiñones, E. Berger, G. Bernat, and F. Cazorla, "Using Randomized Caches in Probabilistic Real-Time Systems", ECRTS 2009: Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems, Dublin, Ireland, July 1-3, 2009.

[RBW+06] A Definition and Classification of Timing Anomalies. Jan Reineke, Bjorn Wachter, Stephan Thesing, Reinhard Wilhelm, Ilia Polian, Jochen Eisinger, and Bernd Becker. 6th Intl WORKSHOP ON WCET ANALYSIS Dresden, Germany, July 4, 2006.

[R] The R project for statistical computing - www.r-project.org/

[RESP] G. Beltrame, C. Bolchini, L. Fossati, A. Miele, D. Sciuto. "ReSP: A Non-Intrusive Transaction-Level Reflective MPSoC Simulation Platform for Design Space Exploration". ASP-DAC'08, Seoul, South Korea

[SAFE] K. Hoyme and K. Driscoll. Safebus TM. IEEE Aerospace Electronics and System Magazine, pages 34-39, Mar 1993.

[SAS] Bussiness Analytics - www.sas.com

[Ski01] Steven Skiena “Calculated bets: computers, gambling, and mathematical modeling to win”, Cambridge University Press, 2001.

[SOCLIB] SoCLib: System On-Chip Library Simulator framework. www.soclib.fr

[SPAD] www.spadsoft.com

[SPSS] http://www.spss.com/

[TDS+95] T.S. Tia, Z. Deng, M. Shankar, M. Storch, J. Sun, L.C. Wu, and J.S Liu. Probabilistic performance guarantee for real-time tasks with varying computation times. In IEEE Real-Time and Embedded Technology and Applications Symposium, 1995.

[Werm03] M.Werman. Fast Convolution. In the Proceedings of the 11th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG03), 2003.

[WIN] The WinBugs project - http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/contents.shtml

[Wo07] W. Wolf. High-Performance Embedded Computing. Morgan Kaufmann, 2007

[ZHL+02] H. Zhu, J.P. Hansen, J.P. Lehoczky and R. Rajkumar. Optimal partitioning for Quantized EDF Scheduling. In the Proceedings of the 23rd IEEE Real-Time Systems Symposium (RTSS02), pp. 202-213,2002.