Method and Tools for On-board Software Engineering
Proposal for TEC-SW/09-301/SoW
Tasks 3 & 4: Revised Technical Proposal

Dr. Andrew Butterfield
Lero/Trinity College Dublin
October 13, 2011

1 Introduction

This document describes a revised proposal for Tasks 3 & 4 in ESA Request for Tender RFQ/3-13168/10/NL/LvH, revised after the meeting held on 9th/10th February 2011 at ESTEC.

The activities to be revised are as follows:

**Task 3** Generic Partitioning Kernel Software Reference Specification

**Task 4** Secure Separation Kernel Verification

The key issue that was clarified at the meeting is that the purpose of Tasks 3 and 4 is to perform research into a formal verification/validation process that would fit with ESA’s software development methodologies, by developing a reference specification, identifying a formalism and process, and running a pilot feasibility study on sample code. The actual code to be subjected to verification in this trial is for example purposes only and would not be the code that would be used or developed for an ESA mission. The latter would require both the proposed formal verification process and corresponding code to go through ESA qualification, which is a major undertaking that would only occur if a future mission decided it requires such verified code.
The goal of tasks 3 & 4 can now summarized as a research activity into a process that would be required in order to obtain EAL5+ certification (or equivalent) for a time/space-separating partitioning kernel. The implementation language would be C, written according to ESA’s coding standards, and the current hardware target would be the Leon 3 processor. ESA already has 3 such kernels available to it and one of those, appropriately modified, would be the eventual target of a full-scale verification effort — ESA has no desire to acquire a 4th microkernel.

Before looking at Tasks 3 & 4 in more detail, we give responses to top-level questions raised in the feedback given by ESTEC on Day 2 of the meeting:

- **Can a EAL5+ evaluation be achieved without a full machine checked verification?**
  
The formality requirements for EAL5+ are as follows:

  - **EAL5** Formal Security Policy Model, Semi-Formal Functional Specification; High-Level design; and Representation Correspondence.
  - **EAL6** EAL5 plus Semi-Formal Low-Level Design.
  - **EAL7** EAL6 plus Formal Functional Specification; High-Level design; and Representation Correspondence.

  The verification proofs occur as part of Representation Correspondence and are only required between levels that have formal descriptions.

- **What is meant by semi-formally verified (EAL5 & 6)? Mix of functional testing and formal analysis**
  
The EAL levels build on top of one another, adding successively more rigorous degrees of verification and validation. Increasing degrees of formality are introduced in stages, but the prior requirements for functional testing still remain, down to the implementation representation level (code), with complete independent confirmation of test results. In any case, right up to EAL7, formal verification is only required between the architecture and high-level design levels. Low level design can be semi-formal. From CC [1, Part 1] we get:

  - **formal** expressed in a restricted syntax language with defined semantics based on well-established mathematical concepts.
  - **semi-formal** expressed in a restricted syntax language with defined semantics.
• *Medium Robustness version of SKPP is under definition. Has it been formally issued? Can we use it?*

The only OS profiles currently US government approved are the High Robustness SKPP, and a general EAL2+ general purpose OS Profile. The most recent information from the NIAP CCEVS website is that the SKPP is currently being “sunsetted” as it is seen to have problems with more complex platforms (i.e. so called “commodity workstations”). The issue seems to be complex interactions between commodity components. To some extent this is not relevant to this work, as we would not be targeting such commodity platforms, but it does mean that SKPP will not be available to us as an official certification route. The Protection Profile for Partitioning Kernels at High Robustness (PPPK, ²) sets a standard that is basically EAL5, but with certain parts raised to EAL6 or EAL7 standards, most notably the software development parts. In particular it raises all the requirements for formal verification up to EAL7 standard.

• *Is there an overlap where SKPP verifies non-security properties?*

MILS principles should mean that security level verification will not itself involve verification of lower levels. But EAL5 does require functional testing down to low-level design, and validation of the covert channel analysis, which will require looking at lower levels. It seems that there should be an overlap.

## 2 Activities

### 2.1 Task 3: Generic Partitioning Kernel Software Reference Specification

The focus of task three is on the details of a partitioning kernel in the first instance, followed by a look at the elements to make this into a separation kernel, in keeping with the MILS philosophy of separation of concerns, in order to avoid excessive verification complexity. This is accompanied by a survey of appropriate formal verification processes.

Task 3 has a duration of 6 months.

R-3.1 Define Reference Specification for partitioning kernel [D03]

¹ [http://www.niap-ccevs.org/pp/pp_skpp_hr_v1.03/](http://www.niap-ccevs.org/pp/pp_skpp_hr_v1.03/)
R-3.2 Assess reference specification [D03]

R-3.3 Extend reference specification to separation kernel [D03].

R-3.4 Establish a formal verification process: approaches; feasibility; cost [D-04].

Task 3 will provide the following deliverables:

[D03] Reference Specification for Partitioning and Separation Kernels

[D04] Report on proposed Separation Kernel Verification Process

[D05] Presentation for ESTEC of material and results in D03,D04.

2.1.1 Sub-Task R-3.1

This sub-task concerns itself with assembling a reference specification for ESA use, based on a variety of existing published specifications, such as AIR-2, Xtratum and L4. The latter two are publicly available to some degree — Xtratum publish theirs as user and reference manuals, whilst the sel4 kernel from NICTA can be found as a published Haskell model. However, the AIR-2 kernel is based on the ARINC 653 standard, which is proprietary, at a current cost of $425. Also Part 2 of of the Common Criteria will be used to guide the production of requirements.

The reference specification is required to cover material from ECSS-E-40c as follows:

Technical Specification

Architectural Design Document: the overall conceptual structure of the microkernel, structured according to MILS principles, modified as appropriate to cater for space software requirements.

SW/SW Interface Control Document: The API of kernel services, as seen by software running in any partition.

HW/SW Interface Control Document: The hardware support for the kernel, made available through an appropriate API specification. Here it is anticipated that the 32-bit LEON architecture, developed by ESTEC and Aeroflex Gaisler, will be used.
A key point to be borne in mind during this subtask is to ensure adherence to MILS principles, to ensure that the subsequent verification of same is feasible.

We see the key output of subtask 3.1 as being a draft reference specification addressing the aspects raised above, suitable for feeding into deliverable D03.

2.1.2 Sub-Task R-3.2

This subtask is mainly an exercise in two things: (i) a review of the literature beyond the three kernels mentioned above to uncover other services that might be of interest to ESTEC, and (ii) consultation with ESTEC on space-specific aspects that may need to be covered.

We don’t view this activity as starting once subtask 3.1 has terminated, but rather as running in parallel, as the reference specification is assembled. One particular reason for this is that any additional services proposed here may have an impact on other aspects, so some interplay between the two subtasks is to be expected. Again, the output of this subtask will feed deliverable D03.

2.1.3 Sub-Task R-3.3

This subtask builds on the results of the previous two, which have resulted in the specification of a Partitioning Microkernel. It proceeds by adding in the security properties, concerned with maintaining process separation, to result in the specification of a Separation Microkernel. This involves a review of the SKPP for high/medium robustness, to identify the security properties, in order to add the relevant function requirements from the protection profiles into the reference specification.

These security properties give assurance regarding the effective separation of different applications. From the separation microkernel’s point of view, all application programs within the partitions are viewed as hostile, and it guarantees that any bad behaviour is kept within a partition. This means that a late change to any partitioned software does not have to be verified in the context of all the other applications being deployed.

This subtask starts once the previous two have largely completed, and merges the results of all three subtasks into D03.
It should be reiterated here, that the SKPP is being “sunsetted” as it has proved to be to difficult to achieve, in its intended application area, namely that of security properties of operating systems built on general purpose commodity hardware. The issue is that assurance requires knowing about the hardware platform to a degree that is almost impossible in this market sector. The reason that Greenhill managed RTOS certification was because they focussed on specific hardware platforms, rather than trying to cover a wide range. As this work intends to focus on the Leon architecture, we should not have any difficulty in this regard, so the continued use of the SKPP as a means of structuring our security requirements and guiding their assurance is viable.

2.1.4 Sub-Task R-3.4

We note that code generation is provide a vehicle for functional verification (testing)

CC 2 covers the details of the security functional requirements, while CC 3 deals with how to assure security is achieved, some of which discussed how to verify security functions

Different Approaches to EAL Cerification (1) describe what CC3 requires.

Common Criteria part II (CC2) describes the elements that should make up the security functional requirements, and so its prime focus is on the content of any specification documents. Part III (CC3), by contrast, is concerned with assurance, i.e. the methodology by which the security of the system is evaluated. For our purposes, the two most relevant Assurance Classes, are Development (ADV) and Tests (ATE). The Tests class [1, part 3, p158] has four families: Coverage (ATE_COV); Depth (ATE_DPT); Functional tests (ATE_FUN); and Independent testing (ATE_IND), all of which much take place, regardless of how much formal verification is done in the Development class. The Development class [1, part 3, p101] has seven families:

ADV_FSP Functional Specification

ADV_HLD High-level design

ADV_IMP Implementation representation

\footnote{NIAP will no longer accept any new applications for evaluation under SKPP}
ADV_INT  TSF internals
ADV_LLD  Low-level design
ADV_RCR  Representation correspondance
ADV_SPM  Security policy modelling

The following table shows, for EAL5–7, the degree of formality required by each development family [1, part 3, p64]:

<table>
<thead>
<tr>
<th>Level</th>
<th>semi-formal</th>
<th>formal</th>
<th>proof</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAL5</td>
<td>FSP HLD RCR</td>
<td>SPM</td>
<td></td>
</tr>
<tr>
<td>EAL6</td>
<td>FSP HLD RCR LLD</td>
<td>SPM</td>
<td></td>
</tr>
<tr>
<td>EAL7</td>
<td>LLD</td>
<td>FSP HLD RCR LLD SPM</td>
<td>RCR</td>
</tr>
<tr>
<td>PPPK EAL5+</td>
<td>LLD</td>
<td>FSP HLD RCR LLD SPM</td>
<td>RCR</td>
</tr>
</tbody>
</table>

In summary, the mapping of SKPP verification to tests and formal analysis is described in CC3 and we would take this as a baseline. We will also use the “class_family” naming system as described above to allow us to reference the requirements. The PPKP reccomendation for high-robustness is usually described as EAL5+, the “+” resulting from the fact that the ADV requirements (among others) are raised to EAL7 standard.

Current Approaches to Formal Verification (2) talk about the three known approaches: NICTA et al

Greenhills INTEGRITY RTOS Greenhills have obtained, and continue to obtain, SKPP EAL6+ High Robustness certification for their real-time operating system. They built their system from the ground up with certification in mind from the start, and the current certification work focusses on software upgrades and the targeting of new hardware platforms for their customers (many PowerPC based).

NICTA sel4 The sel4 verification by NICTA [3] used Haskell as a modelling language to provide (i) a executable prototype and (ii) a means of auto-generating the abstract specification. The C code was generated by hand and the verification was performed using Isabelle/HOL, based on a formal semantics of their C subset, and the use of Hoare-triple based reasoning.
PikeOS Work in the Verisoft project looked at using Microsoft’s Verifying C Compiler (VCC) to prove correctness of the PikeOS operating system. The emphasis is on hand-annotating the C code with specification details and verification is carried out using the Z3 theorem prover. The emphasis in this work is on the lower levels, from C code down to the resulting assembler.

Critical Systems A feasibility study was carried out by Critical Systems [4] using Atelier-B to model and verify some security properties, relying on the code-generator to provide the code under study.

OpenComRTOS The OpenCOMRTOS project looked at developing a real-time o/s kernel starting with a formal model and specification [5]. They used the temporal logic of actions (TLA) as their formalism, and used the TLA+ Toolbox, for verification.

The Greenhills approach is proprietary and closed to us. Of the remaining three approaches, only the Critical System explicitly mentions CC and SKPP, but their work appears to be a small feasibility study over automatically generated code. The NICTA study is the most open, in terms of published details about both the methodology and the artifacts involved. While they do not discuss CC/SKPP, their approach looks like it would be compatible with such an evaluation framework.

Sub-Task R-3.4 Approach (3) say what we would do

In the Common Criteria only EAL7 requires formal proof, albeit between all levels from security policy down to low-level design. However the Greenhills work, certified to EAL6+ (High Robustness) has involved formal proof, and some of these may be attributed to extra requirements from the high-robustness part, which goes beyond the basic SKPP/EAL definition. Ee propose that this work seeks to scope out a development process that will achieve a comparable level of certification. We know this is feasible, in that Greenhills have achieved such certification, and the work done by NICTA gives us access to some details of how they proceeded with their verification.

We can see that sub-task 3.4 needs to:

- Identify the formal notations used to describe the various development levels: security model; specification; high- and low-level design (code).
• Identify the appropriate proof tools to verify the required representation correspondence between each formally described level.

• Identify a modelling approach coupled with a code-generation mechanism that can be used to provide a strawman code-base in order to explore the verification process.

In all of the above, the ideal outcome would be one that minimised the diversity of both formal notations and proof tools. All decisions should be guided by the fact that the implementation language be C, conforming to the MISRA standard [6], and that the hardware platform being targeted is the LEON-3 processor. Another key concern, given that any future formal verification process will have to go through ESTEC qualification, is how to judge the soundness of any proof tool being evaluated. The importance of this aspect of any reasoning tool will tend to favour logics implemented on a small trusted proof kernel, like HOL or CoQ, rather than those based on large complex (fast?) decision procedures, such as PVS.

**Verification Cost**  These estimates will be prepared based on published figures in the literature for all the methods that are considered, and also through discussion with potential industrial collaborators familiar will labour costs associated with the kinds of skill levels that would be required.

All the above issues will be compiled into a report that will comprise D04.
2.2 Task 4: Secure Separation Kernel Verification

The focus of Task 4 is to take the decisions made in Task 3 and bring them to the point where a feasibility study on an actual formal verification task can be undertaken. Formalisms and modelling languages, with associated tool support, will be chosen, and these then used to obtain the formal abstract specification and the kernel code that will be inputs into the verification feasibility study.

Task 4 has as inputs the following deliverables from Task 3:

[D03] Reference Specification for Partitioning and Separation Kernels

[D04] Report on proposed Separation Kernel Verification Process

Task 4 has a duration of 12 months.

R-4.1 Evaluate formalisms for suitability [SW04,D06]

R-4.2 Evaluate modelling languages [SW05,D06]

R-4.3 Generate abstract specification [D-07]

R-4.4 Apply process: produce abstract specification [SW02]; model [SW03?,SW05]; verification attempt; and assess [D08].

Task 4 will provide the following deliverables:


[D07] An Abstract Specification of the Separation Kernel using language(s) recommended in D06.

[D08] Report assessing how the process defined (loosely) in D04 will be implemented, given the recommendations made in D06, and the languages chosen to produce D07, as well as results from an initial trial run of the process.

[SW02] The source material in the chosen specification language(s) of the Abstract specification described in D07.
Seperation Kernel program code: either autogenerated from the Abstract Specification (SW02) if this proves feasible, or hand-coded, at least to the extent that the trial run leading to D08 becomes practical.

Toolset for the chosen formal language and method

Toolset for the modelling language and autocode generation, should this functionality not be supported by the tools in SW04.

The precise nature of the activities in Task 4 will depend on evaluations and decisions made in Task 3, and most of the key issues have been aired there. Hence the material here is shorter.

2.2.1 Sub-Task R-4.1

Our attention has already been drawn to three contemporary tools-based verification technologies: NICTA’s use of Haskell/Isabelle/HOL, Critical Systems use of Atelier-B and Sysgo’s work with Microsofts’ Verifying C Compiler. We are also aware of work [7] which looks at the use of the Z/Eves theorem prover to machine-prove the Z model of a kernel described in RD-11 [8]. One of those authors has also done work on the Xenon Hypervisor [9] using Z, CSP and Circus [10]. In addition, a book has just been published [5] that describes the formal development of a network-centric RTOS, using the TLA+/TLC formalism and toolset [11]. We would explore these and other technologies such as Coq, PVS, other B-related tools (the Rodin toolset), looking at suitability, useability, and availability, both to us for the duration of the work, and for ESA for subsequent use. This subtask may be quite short if a clear front runner emerges during Task 3 and the ensuing discussions with ESTEC, but we need to keep some time aside for this subtask, because we may need to experiment with the tools ourselves to get a feel for their usability. Also important to keep in mind is the need to have regard for the soundness arguments that can be made for the various tools with a view to future qualification by ESTEC of same.

We note the following observations about the availability of tools for the approaches mentioned above:

NICTA seL4 used Haskell, Isabelle/HOL, stock compiler, all freely available (open source)
**Sysgo, PikeOS** using Microsofts’ Verifying C Compiler, available for free, but only for *non-commercial* use.

**Critical Systems** using Atelier-B, freely available, with fees for maintenance

**Freitas et al, Iain Craig’s Z Kernel** using **Z** and **Z/Eves**, the latter currently being unavailable due to licensing issues involving the Canadian Govt.

**Freitas et al, Xenon Hypervisor** using **Z**, **CSP** and **Circus**. The tools used were **Z/Eves**, and the Circus CZT Tools—the latter are freely available but are still very much at a prototype stage.

**openComRTOS** used **TLA+/TLC**, which is available from Microsoft under their Research License Agreement, which stipulates non-commercial use, or commercially *solely for (your own) internal use*

It should be noted that software from Microsoft mentioned above for non-commercial use only, may be obtained for commercial purposes, but this will need to be discussed with Microsoft.

### 2.2.2 Sub-Task R-4.2

Formal methods tools that autogenerate code are an obvious candidate here (like Atelier-B). However if some other modelling language produces better code, particularly at low-level, then the difficulty of finding a formal semantics for the generated code arises. The key here is to have generators that target MISRA-C[6], or similar, as these standards describe a subset of C that is simple enough for formal reasoning to be tractable. In any case, in order to perform verification, the tool suite used will need to support the formal semantics of the C subset used.

We see subtasks 4.1 and 4.2 running concurrently as there is an obvious interplay between the two, particularly if a formalism that supports code generation is selected, subject to the code so generated being of adequate quality with a formalism that can handle the hardware-level properties that are needed to guarantee security.
2.2.3 Sub-Task R-4.3

Given that the formalisms, modelling languages and tool have been identified, we then proceed to the production of the abstract specification. In effect we encode the reference specification in the formal specification language that has been identified. In this activity we will mirror the way the reference specification has been structured according to MILS guidelines, because these were specifically designed as an architecture to ease the necessary verification effort.

2.2.4 Sub-Task R-4.4

Finally, we apply the process established in WP3.1 to formally verify the separation kernel. The objective is not to complete the verification, because that is not feasible with these resources, but rather to get a feel for the feasibility, and cost of a full-scale complete verification, keeping a close eye on the kinds of guarantees that are required for space missions.

The overall flow envisaged in Tasks 3 and 4 is as follows:

1. Produce a Reference Specification (informal)
2. Produce an Abstract Specification (formal)
3. Produce Kernel Implementation code (formal, given formal semantics of code)
   Either by:
   
   (a) auto-generating it from the Abstract Specification (which may obviated the need for step 4 below — effectively it is subsumed into step 2 above).
   
   (b) construct a tool-based Specification Model and use that to automatically generate the code
   
   (c) produce the code by hand.

4. Prove that Implementation satisfies the Abstract Specification

The Critical Systems described approach involves Steps 2 and 3(a), whilst NICTA used prototype code in Haskell to develop the abstract specification, and then used steps 3(c) and 4. The Sysgo approach seems to start with step 3(c) and then perform step 4 as a means to generate the result of step 2.
The NICTA approach, or a suitable variant of it, could be reworked to fit the flow described above, and the Critical Systems approach is also a good fit. It is quite possible that we may be able to take the best aspects of both approaches and fuse them together — using Haskell to allow us to prototype a kernel whose specification is then captured using the Atelier-B tools.

3 Study Plan

3.1 Task 3: Generic Partitioning Kernel Software Reference Specification

Task 3 consists of the following subtasks:

- Subtask 3.1 Initial partitioning kernel reference specification
- Subtask 3.2 Specification assessment
- Subtask 3.3 Separation kernel extension to reference specification
- Subtask 3.4 Process for formal verification

As shown in Figure[1] subtasks 3.1, 3.2 and 3.3 run in sequence, progressing towards the issuing of D03. Subtask 3.4 runs in parallel with the other three, ramping up as the growing specification makes the kind of verification required become more apparent.

3.1.1 Deliverables

Task 3 will provide the following deliverables:

[D03] Reference Specification for Partitioning and Separation Kernels
[D04] Report on proposed Separation Kernel Verification Process
[D05] Presentation for ESTEC of material and results in D03,D04.
Figure 1: Workflow for Tasks 3 & 4
3.2 Task 4: Secure Separation Kernel Verification

Task 4 consists of the following subtasks:

- Subtask 4.1 Evaluation of formal languages and methods
- Subtask 4.2 Assess and select modelling language
- Subtask 4.3 Abstract Specification
- Subtask 4.4 Formal verification feasibility study

As shown in Figure 1, Subtasks 4.1 and 4.2 run in parallel as choices need to be made in both subtasks for tools that can work together in the desired process. The output of 4.2 may be subsumed by choices made in 4.1. Tasks 4.3 and 4.4 run sequentially once 4.1 and 4.2 are complete, with some overlap, as results from 4.4 may require changes to the abstract specification produced in 4.3.

Task 4 will provide the following deliverables:


[D07] An Abstract Specification of the Separation Kernel using language(s) recommended in D06.

[D08] Report assessing how the process defined (loosely) in D04 will be implemented, given the recommendations made in D06, and the languages chosen to produce D07, as well as results from an initial trial run of the process.

[SW02] The source material in the chosen specification language(s) of the Abstract specification described in D07

[SW03] Separation Kernel program code: either auto-generated from the Abstract Specification (SW02) if this proves feasible, or hand-coded, at least to the extent that the trial run leading to D08 becomes practical.

[SW04] Toolset for the chosen formal language and method

[SW05] Toolset for the modelling language and automatic code-generation, should this functionality not be supported by the tools in SW04.
4 Statement of Compliance

The work described in this document is compliant with the Statement of Work, with the exception of the deviations in Table 1. This table contains clarifications and elaborations of the tasks and subtasks where deviations occur (or may be interpreted to occur).

Table 1: Matrix statement of revised compliance of Tasks 3&4

<table>
<thead>
<tr>
<th>Rqrmnt</th>
<th>Task</th>
<th>Proposed Deviation/Elaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-3.1</td>
<td>3.1</td>
<td>compliant</td>
</tr>
<tr>
<td>R-3.2</td>
<td>3.2</td>
<td>compliant</td>
</tr>
<tr>
<td>R-3.3</td>
<td>3.3</td>
<td>compliant</td>
</tr>
<tr>
<td>R-3.4</td>
<td>3.4</td>
<td>compliant</td>
</tr>
<tr>
<td>R-4.1</td>
<td>4.1</td>
<td>compliant</td>
</tr>
<tr>
<td>R-4.2</td>
<td>4.2</td>
<td>compliant</td>
</tr>
<tr>
<td>R-4.3</td>
<td>4.3</td>
<td>compliant</td>
</tr>
<tr>
<td>R-4.4</td>
<td>4.4</td>
<td>compliant</td>
</tr>
</tbody>
</table>

References


17


URL [www.misra.com](http://www.misra.com)

URL [http://dx.doi.org/10.1007/978-3-642-14808-8](http://dx.doi.org/10.1007/978-3-642-14808-8)


URL [http://dx.doi.org/10.1007/s10009-011-0195-9](http://dx.doi.org/10.1007/s10009-011-0195-9)

URL [http://dx.doi.org/10.1007/s00165-007-0052-5](http://dx.doi.org/10.1007/s00165-007-0052-5)