

ULRR

Towards formal verification of separation microkernel

Item Type	Meetings and Proceedings
Authors	Butterfield, Andrew;Sanán, David;Hinchey, Mike
Citation	DASIA 2013, annual Eurospace conference;
Download date	2026-03-16 21:38:01
Item License	https://creativecommons.org/licenses/by-nc-sa/1.0/
Link to Item	https://hdl.handle.net/10344/3364

TOWARDS FORMAL VERIFICATION OF A SEPARATION MICROKERNEL

Andrew Butterfield¹, David Sanán¹, and Mike Hinchey²

¹*Lero. Trinity College Dublin*

²*Lero. University of Limerick*

ABSTRACT

The best approach to verifying an IMA separation kernel is to use a (fixed) time-space partitioning kernel with a multiple independent levels of separation (MILS) architecture. We describe an activity that explores the cost and feasibility of doing a formal verification of such a kernel to the Common Criteria (CC) levels mandated by the Separation Kernel Protection Profile (SKPP). We are developing a Reference Specification of such a kernel, and are using higher-order logic (HOL) to construct formal models of this specification and key separation properties. We then plan to do a dry run of part of a formal proof of those properties using the Isabelle/HOL theorem prover.

1. INTRODUCTION

The concept of separation kernel was introduced by Rushby (Rushby 1981) as a design approach to formally verify complex security kernels. A separation kernel can be considered as a kind of security kernel that creates a virtual distributed environment so each application runs in an environment isolated from other applications also running in the system. In that way it can be considered that applications are running on separate machines and the set of them comprises the virtual distributed environment, in which they can only communicate with each other through the communications channels that the kernel provides.

A separation kernel is used to obtain high assurance in critical systems, not only in those where

users access makes it necessary to ensure a secure environment, but also in embedded systems where it is necessary to have safe process execution and communication. Although the concept of separation helps to design secure and safe systems, kernel complexity results in design and implementation errors so that functional and security properties do not hold. In order to ensure systems correctness, formal methods techniques have become widely used in what is known as formal verification. Formal verification uses theorem proving to check that a formal model of a system satisfies a set of properties given in a formal specification, normally described in some kind of logic. The most well known formal verification approaches are model checking (Clarke et al. 2001) and deductive verification. The former is usually fully automatic but does not scale well for big systems; the latter requires that the user has a better knowledge of the system to verify, but it scales to large systems.

To that aim, within the project *Methods and Tools for On-Board Software Engineering*¹, we are working on the functional and security requirements specification, modelling, and verification of a separation microkernel according to the requirements baseline for the IMA-SP platform (de Ferluc 2012) being developed as part of ESTEC's Savoir-IMA activity. It is a feasibility study into what would be required to verify such a kernel to EAL Level 5+, as described by the Common Criteria (CC) framework (Criteria 2005) and the Separation Kernel Protection Profile (SKPP) (Directorate 2007).

The requirements specification describes the desired kernel behaviour to handle partitions, inter-

¹funded by ESTEC CONTRACT No. 4000106016

partition communication, memory management, devices handling, trap handling, and fault detection, isolation, and recovery. The kernel architecture as described by this specification, including kernel structure, data structures, and functions provided by the kernel are to be modelled in a high level formalism. To be able to carry out the verification the model also needs to include a formal representation of the underlying hardware, of the subset of C used, as well as the assembly instructions use for low-level hardware access. As regards the hardware model, we consider a reliable architecture so only the relevant components for the verification need to be modelled, in particular those components like the MMU where inappropriate setup and use can lead to security violations.

It is worth to highlight that, in order to achieve a full kernel verification, the verification has to be performed at several levels of abstraction that capture all the implementation details as shown in the PSOS project (Feiertag & Neumann 1979), one of the initial approaches for kernel verification. For instance, the correctness of the specification over a high level abstraction model of the kernel does not ensure the correctness of the implementation and the assembly code. Therefore for each architectural level a new layer of abstraction capturing all the details has to be provided and verified, which increases the cost of verification. However this cost can be reduced using techniques like formal refinement. Formal refinement verifies that the behaviour of a lower (closer to implementation) layer satisfies the requirements of a higher (closer to specification) layer, typically using a concept known as forward simulation. Forward simulation applies a refinement relation over states belonging to different abstraction layers in such a way that one layer refines the other one if the successors of two states related by the refinement relation belongs also to the refinement relation, and the properties that hold in the refined layer also hold in the refiner layer.

The ongoing work will develop a high level model that catches the significant behaviour of the separation kernel requirements specification and similarly for the underlying hardware (e.g., memory, MMU, traps, or general registers). Then functional and separation requirements for a separation kernel will be verified over the abstract model. One important aspect to consider in

the development process is the scalability of the verification. To that aim the models will be constructed with a high degree of orthogonality and independence, so small changes in requirements or in the implementation will have minimal effect on the verification of unaltered parts of the system.

2. BACKGROUND: SEPARATION KERNEL CONCEPTS

The task of a separation kernel is to create an environment which is indistinguishable from that provided by a physically distributed system: it must appear as if each regime is a separate, isolated machine and that information can only flow from one machine to another along known external communications lines. (Rushby 1981)

Introduced by Rushby (Rushby 1981), separation concepts seek to simplify the design and verification of a high assurance security model. Previous attempts in the design of such security models failed to set up a architecture simple enough to make a sound verification feasible. Basically, there are two reasons for this. On the one hand the presence of trusted processes made the kernel verification complex. Some clear examples of trusted processes can be found in those systems including printer or authentication services, where such services needed to be considered as trusted processes, so becoming part of the kernel, and therefore increasing the verification complexity and effort. On the other hand, the lack of certainty in their security models meant they were not precise enough to capture the basic characteristics of a security kernel to the degree needed to provide a basis for sound verification.

Separation kernels are intended to solve these problems by considering the system as a physical distributed system where the processes running in the kernel are independent machines in that distributed system, and the secure kernel is the mechanism to provide a virtual machine that shows a unique view of the hardware for each process, and to enforce logical separation between them. In order to perform logical separation two aspects are considered: both spatial and temporal separation of applications. On the one hand, spatial separation assigns different resources to each application, which results in ap-

plication isolation, whereas on the other hand, temporal separation separates application activities in time. In effect, both temporal and spatial separation concepts result in the notion of a partition as an independent execution environment for each process.

Although pure separated environments, that is environments where partitions do not share any information flow among them, are simple and therefore can be easily verified (Rushby 1981), actual systems need inter-partition communication. Since information flows break the spatial separation principle, the separation kernel protection profile (SKPP) (Directorate 2007) establishes a Partition Information Flow Policy (PIFP) that the kernel enforces so that only appropriate flow of information is allowed. So, partitions can only communicate with others by using flows explicitly permitted by the PIFP.

Going back to the problems related with trusted processes in previous secure kernels, separation kernels allow us to construct hierarchical architectures where those services provided by trusted processes can be implemented as top layers of the separation kernel. This allows trusted processes that use separation kernel services to develop the require functionality without augmenting the kernel complexity, making the trusted process and kernel verification independent of each other. So, the MILS architecture (Alves-Foss et al. 2006) provides applications layers with mechanisms to control, manage and enforce application level security policies.

The concept of separation was redefined by Alves-Foss et al. in (Alves-Foss et al. 2002) to consider the concepts of virtual machines for separation used by Rushby and of environment observability independence proposed by Hardin et al. in (Matthew et al. 2003): *The behaviour and performance of software in one partition must be unaffected by the software in other partitions except through communication through authorized communications channels.*

3. STATE OF THE ART

The use of formal methods for kernel verification dates back to the end of the 70's. Although formal verification tools were not mature

enough, and no full implementation proofs were given, UCLA Secure Unix (Popek et al. 1979), PSOS (Feiertag & Neumann 1979), and KIT (Bevier 1989) were the first steps at operating system formal verification. Despite of the lack of proofs, these projects settled the methodologies to achieve full kernel verification such as a layered architecture, and refinement based proofs where the correctness of abstract layers allows the inference of the correctness of the concrete layers.

Later in the 2000s, the verification technology was mature enough to support the verification of a new generation of microkernels, such as L4. So, VFiasco (Hohmuth & Tews 2005), targeting the kernel Fiasco belonging to the L4 family, became successful in verifying some list properties. Also important to highlight is the Verisoft activity (Alkassar et al. 2008) that performed the pervasive verification of a system from the application layer, going through a microkernel, and a compiler, to the hardware. The importance of separation/partitioned micro-kernels has motivated up new efforts in the verification of new secure kernels like seL4, or PikeOS, based on the separation paradigm and with a MILS-compliant architecture. The seL4.verified kernel (Boyton 2009) targets the high-assurance kernel market. It uses a three step refinement based proof verification to verify an abstract specification, an executable specification, and finally a C implementation. They initially aimed to verify purely functional requirements, although the properties verified do cover security-related aspects like buffer overflows, or memory leaks. The Verisoft project has been followed by Verisoft XT, which targets different OS, one among them being the separation micro-kernel PikeOS. Verisoft XT aims at the verification of PikeOS functional requirements while being in non-concurrent mode by disabling the interrupts.

With regard to the separation kernel proofs, Rushby (Rushby 1981) initially described an approach to the separation problem for non-interfering systems, and later on for partitioned systems sharing communications channels (John 1999). In 2002, Alves-Foss et al. (Alves-Foss et al. 2002) give a proof of separation for a formal model over a machine model based on a stack architecture, the Idaho Partitioning Machine. Finally, it is worth pointing out some recent work in Verisoft XT and seL4.verified that focus on

separation properties. So, VerisoftXT in (Baumann et al. 2011) verify partition memory separation with strict separation, and no communication among partitions. The sel4.verified project recently proved, in (Murray et al. 2013) an information flow non-interference property, claiming that sel4 can be considered as a separation kernel.

4. MTOBSE

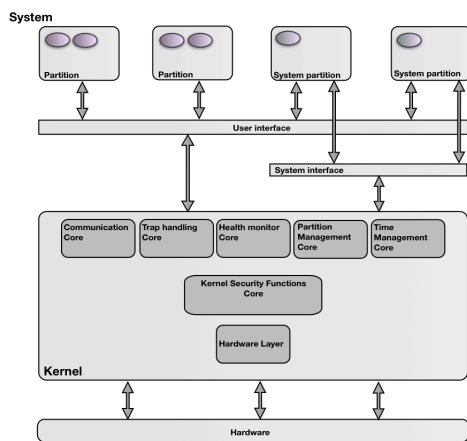


Figure 1. Separation Kernel Design

The project *Methods and Tools for On-Board Software Engineering* (MTOBSE), in collaboration with the European Space Agency (ESA), intends to address key issues in the on-board software domain, in particular the formalization of Time and Space Partitioning kernels, by applying software engineering methodologies. To that aim MTOBSE defines a set of activities which include the generation of a reference specification for a separation micro-kernel, the evaluation of suitable tools and formalisms for its formal verification, the modelling of a separation micro-kernel in the selected formalism based on the reference specification firstly generated, to finally end up with a verification trial and feasibility study.

4.1. Reference Specification

The developed reference specification consists of software requirements (Butterfield & Sanan

2013c), interface requirements (Butterfield & Sanan 2013b), and the architectural design (Butterfield & Sanan 2013a). Both the software requirements and the interface requirements are specified following the requirements baseline for the IMA-SP platform (de Ferluc 2012), and ESA's own suggestions. Based on the specified requirements, the architectural design provides the data structures, component internal interfaces, and component functionalities.

The main sources from which the software requirements are drawn are the Arinc-653 standard specification (ARINC 2005) for functional requirements (partitioning aspects) and the Separation Kernel Protection Profile (SKPP) (Directorate 2007) for security requirements (separation aspects). Although the provided reference specification is guided these standards, it does not intend to be compliance with them. So, in the case of the partitioning requirements, although the specification is compliant with partition, scheduling, inter-partition communication, and time requirements, the ones for processes, intra-partition communication, or health monitor are not, since according to ESA the management of those aspects are handled by a partition guest OS or a system partition.

In the case of security and separation requirements this reference specification follows SKPP indications, and it includes adapted requirements tailored to the space environment for audit, user data protection, identification and authentication, security management, protection of the security functions, and resource utilisation.

Figure 4 shows the diagram for the architectural design. It includes the necessary components according to the reference specification, that is: Partition core to manage partitions; Communication core for Inter-partition communication; Time management core to provide partitions with global and local time; Health monitor core to handle software and hardware exceptions not concerned with the kernel security functionality; Trap core to manage the interrupts and devices; and finally the Kernel Security Functions core, which is in charge of security requirements as specified in the SKPP. User and System partitions can access the functionality the kernel provides through the interface these modules implement.

4.2. Formalism and Tool Evaluation

Formal verification involves the modelling of the system and the properties to be verified in some formal language, which later on will be automatically or semi-automatically analysed by a verification tool. The selection of the formalism or the methodology the kernel is going to be modelled with is non-trivial since the language features, as well as the available tools supporting the selected formalism to carry out the verification, determine the feasibility and cost of the verification. For instance, is the chosen formalism expressive enough to model all the kernel and the hardware characteristics? Is it possible to fully formalize the desired properties to be verified? Do any the available tools for that formalism perform a sound and complete verification?

In (Butterfield & Sanan 2012) we give a description of desired features and whether the evaluated formalism and tools fulfill them or not. In a nutshell, the considered features for this evaluation are the following: expressivity of the formalism, how easy it scales for larger models, what semantics are available, is it suitable for modelling of hardware, which are the available tools for the formalism/methodology, which is its scope of verification, track-record of previous verification efforts in the kernel area, and its flexibility to adapt verification requirements.

Considering those features two tools were considered suitable for the verification: the theorem prover Isabelle/HOL (Nipkow et al. 2002), which is based in high order logic and is able to encode different logics like first order logic or separation logic, and VCC (Cohen et al. 2009) an automated assistance for verifying C programs, which uses inline annotation on the source code to which is possible to add pre and post conditions and type invariants as well. Eventually, Isabelle/Hol was selected as the formalism/tool to perform the verification. The main reasons for this selection were the availability of a formal semantics for the C language, and the open source nature of the tools, with the support of a big community of developers.

5. VERIFICATION APPROACH

As mentioned in Section 3, layered architecture based modelling and refinement based verification methodologies have been established as efficient and successful approaches for OS verification. On the one hand layered architectures organise and bind different degrees of system abstractions following a top-bottom design from high abstraction levels to concrete implementations. On the other hand, refinement verification links the different levels of abstraction so the result of the verification performed in the higher levels can be propagated to the lower ones. The greater simplicity of the abstraction level reduces the cost of the verification with regard to the concrete model, therefore the verification of separation properties will be performed in the abstract level. Later on, those verified properties will be inferred in the concrete implementation by means of refinement.

5.1. An Abstract Model for the Separation Kernel

Modelling an abstraction from the functional and security requirements specified in (Butterfield & Sanan 2013c) requires us to consider not only the kernel functionality defining the kernel behaviour, but also to consider those hardware aspects that influence system behaviour (e.g. CPU registers and timers, which are involved in the context switch), as well as those concerning the properties to verify (e.g. the MMU, which enforces spatial separation). It is worth noting that at this stage of the project the hardware is being considered as a reliable component, so it is being modelled at the higher level of abstraction needed to support the required properties.

So far, the abstracted model is made up of more than 27 theories modelling levels from hardware up through kernel types and functional specifications. The hardware is composed of the theories *SparcV8MachineTypes*, *MachineTypes*, *MMU*, and *MachineState*. The theory *SparcV8MachineTypes* models the necessary CPU data types like physical and virtual memory addresses word types, respectively specified as 36 and 32 bit words; virtual pages word types, specified as 32 bit words; or CPU, user, and timer

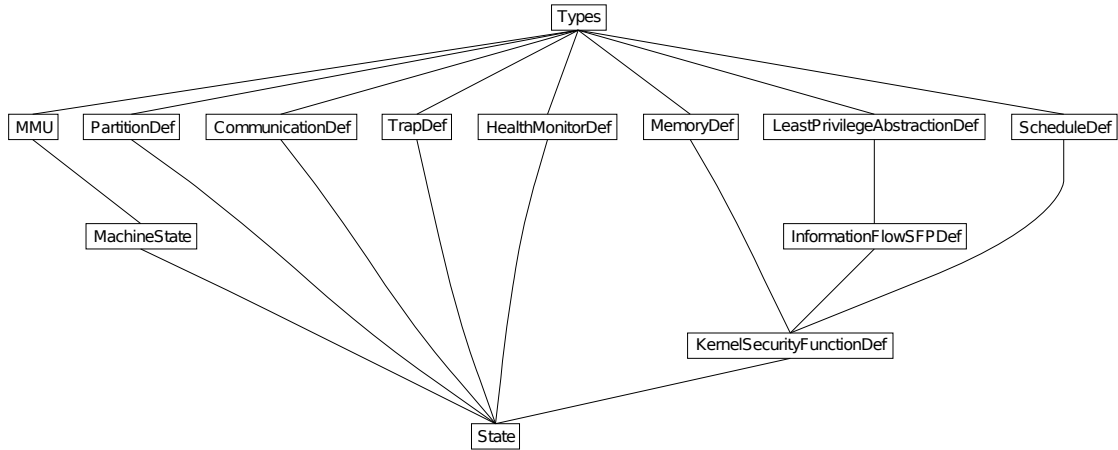


Figure 2. State Definition Model Theory Graph

```

record state=
  p_state :: partitionManagerState
  com_state :: communication_state
  trap_state :: trap_state
  h_m_state :: health_monitor_state
  machine :: machine_state
  ksf :: ksf_state

```

Figure 3. State Definition Model in Isabelle/Hol

registers. This model is specific for the Sparc V8 processor and it is wrapped up by *MachineTypes*, which is used by all the machine-type dependent theories so the model can be easily changed work with other hardware specifications, by modelling the new hardware platform separately, and using *MachineTypes* as an interface. The *MMU* theory models the physical memory and the virtual memory as well, modelling the notion of *contexts* to perform memory separation. Contexts are a dedicated set of pages assigned to each partition, so a partition in a specific context cannot access other memory areas different to the ones assigned to its context. In addition to the main memory, *MMU* also models basic cache functionality. Both the *MMU* and the cache are necessary to define spatial separation properties, and additionally the cache behaviour is important to ensure “sanitisation” properties.

The hardware is accessed by the kernel through a machine state type containing these theories and specified by *MachineState* theory. In addition to

the *MMU*, the CPU and users registers, and the timers, the machine state includes a unspecified model for the unused hardware that can be used in some abstracted functionality.

The kernel model includes the definition of a global kernel state shown in Figure 3 to keep the state of the major components of the kernel defined by the reference specification, as explained in section 4: Partition Management; Communication Management; Trap Management; Health Monitor Management; and the Kernel Security Functions. The Kernel Security Functions also depend on the models for memory, scheduler, and information flow policies to provide spatial and temporal isolation. Figure 2 shows the Isabelle/Hol theories architecture for the kernel state definition and its dependant theories. On the top of the kernel state definition are the theories modelling the security and functionality of the kernel. Hardware and software exceptions are modelled in the *Exception* theory (not shown), which captures software and hardware errors.

5.2. Verifying the Kernel

As mentioned above the higher complexity of the concrete kernel implementation with regard to the abstract model, where data structures and functionalities are simplified to the minimum level necessary to describe the requirements, makes a refinement methodology suitable

to simplify the cost of the verification. So, functional and security requirements are proven in the abstracted model first, and then refinement by means of forward simulation will support the verification of these properties on the concrete kernel implementation.

The verification will be focussed to prove correctness of system calls provided by the kernel, and security properties. Abstract kernel system calls correctness will be proved using pre- and post-conditions, which ensure that system calls meet their functional requirements. On the other hand security properties will be verified using invariants showing that every time a data structure involved in some security property is modified the property is preserved. Another important aspect to consider in the verification is task termination. For that, Isabelle/Hol ensures termination for non-recursive definitions and provides support to prove termination of recursive functions, easing the proof of termination for most of the kernel functions.

Once the abstracted kernel has been successfully verified, we need to obtain the concrete model from the kernel implementation and to apply forward simulation between the abstract model and the concrete model. A concrete kernel model can be obtained using a semantic model of the implementation language over the source code. Although this is indeed a hard process, some related work like the tool developed by NICTA in (Norris 2008) simplify this step. This tool uses a semantic model for C and it translates a program written in C into an Isabelle/Hol model. Thanks to this, a kernel implementation in C can be translated to Isabelle, it only being necessary to provide semantics for some assembly instructions, and perhaps performing some modification in the kernel code to adapt it to the particularities of the semantic model provided by this tool.

Forward simulation is applied by defining a relation R between states belonging to the abstract and the concrete kernels. Briefly, when abstract and concrete versions of an operation α_a, α_c are applied to corresponding states related by R , if the resulting states are also related by R then it can be inferred that the properties verified in the abstract model are also satisfied in the concrete model.

6. FUTURE WORK

So far, an abstract model based on the reference specification is under construction. After the model is finished, and using some available kernel (e.g. Air, PikeOS, or Xtratum) a concrete model from the source code will be obtained using the NICTA tool "*C-to-Isabelle*". As final step, refinement techniques will be applied to the verification in order to assess feasibility and probable cost.

No doubt, this initial verification can be extended to verify further levels, like the compiler or the whole hardware architecture. Moreover, the abstract model can be extended to support other architectures, like multi-core processors, or other microprocessors different to SPARC like ARM.

Finally, one of the biggest concerns of using formal verification to ensure a kernel correctness is the cost of this verification. Even applying methodologies like refinement, which helps to reduce the verification time and cost, the whole verification of a kernel, can take a long time. For instance, in (Klein et al. 2010) the total cost of their functional verification effort was estimated at 25 person-years, without checking separation properties. However, the verification cost can be significantly reduced using a kernel design that does not use complex mechanisms as found in sel4, such as dynamic memory so it holds the maxim of Rushby's separation kernel: keep it the simplest possible.

REFERENCES

- Alkassar, E., Hillebrand, M. A., Leinenbach, D., Schirmer, N. W., & Starostin, A. 2008, in LNCS, Vol. 5295, 2nd IFIP Working Conference on Verified Software: Theories, Tools, and Experiments (VSTTE'08), ed. N. Shankar & J. Woodcock (Springer), 209–224
- Alves-Foss, J., Oman, P. W., Taylor, C., & Harrison, S. 2006, IJES, 2, 239
- Alves-Foss, J., Rinker, B., Benke, M., et al. 2002, The Idaho Partitioning Machine, Tech. rep.
- ARINC. 2005, ARINC Specification 653-2: Avionics Application Software Standard Interface Part 1 - Required Services., Tech. rep., Aeronautical Radio INC

- Baumann, C., Blasum, H., Borner, T., & Tverdyshv, S. 2011, in 1st International Workshop on Architectures and Applications for Mixed-Criticality Systems (AMICS 2011), ed. W. Steiner & R. Obermaisser (Newport Beach, CA, USA: IEEE Computer Society)
- Bevier, W. R. 1989, *Kit: A Study in Operating System Verification*
- Boyton, A. 2009, in *Electronic Notes in Computer Science*, Vol. 254, *Proceedings of the 4th Workshop on Systems Software Verification*, ed. G. Klein, R. Huuck, & B. Schlich (Aachen, Germany: Elsevier), 25–44
- Butterfield, A. & Sanan, D. 2012, *Task 3: A formal verification process: approaches; feasibility; cost*, Tech. Rep. Version 2.0, Lero, ESTEC Contract No. 4000106016
- Butterfield, A. & Sanan, D. 2013a, *mtobse_d03i4_-ADD*, Tech. Rep. Version 4.1, Lero, ESTEC Contract No. 4000106016
- Butterfield, A. & Sanan, D. 2013b, *mtobse_d03i4_-ICD*, Tech. Rep. Version 4.1, Lero, ESTEC Contract No. 4000106016
- Butterfield, A. & Sanan, D. 2013c, *mtobse_d03i4_-SRS*, Tech. Rep. Version 4.1, Lero, ESTEC Contract No. 4000106016
- Clarke, E. M., Grumberg, O., & Peled, D. 2001, *Model checking* (MIT Press), I–XIV, 1–314
- Cohen, E., Dahlweid, M., Hillebrand, M., et al. 2009, in *Lecture Notes in Computer Science*, Vol. 5674, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009*, ed. S. Berghofer, T. Nipkow, C. Urban, & M. Wenzel (Berlin: Springer-Verlag), 23–42
- Criteria, C. 2005, *Common Criteria for Information Technology Security Evaluation*, Version 2.3, Tech. Rep. ISO/IEC 15408:2005, Common Criteria
- de Ferluc, R. 2012, *Report D10 — TSP Services Specification*, IMA-SP/D10 Issue 2.1, ESTEC, contract ESTEC 4000100764
- Directorate, I. A. 2007, *Protection Profile for Separation Kernels in Environments Requiring High Robustness*, Tech. rep., U.S. Government
- Feiertag, R. J. & Neumann, P. G. 1979, in *In Proceedings of the National Computer Conference* (AFIPS Press), 329–334
- Hohmuth, M. & Tews, H. 2005, in *Proceedings of the 2nd ECOOP Workshop on Programming Languages and Operating Systems*
- John, R. 1999, *Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance*, Tech. rep.
- Klein, G., Andronick, J., Elphinstone, K., et al. 2010, *Communications of the ACM (CACM)*, 53, 107
- Matthew, D. G., Wilding, M., & Eet, W. M. V. 2003, in *In Proc. Fourth International Workshop on the ACL2 Theorem Prover and Its Applications*
- Murray, T., Matichuk, D., Brassil, M., et al. 2013, in *IEEE Symposium on Security and Privacy*, San Francisco, CA
- Nipkow, T., Paulson, L. C., & Wenzel, M. 2002, *LNCS*, Vol. 2283, *Isabelle/HOL — A Proof Assistant for Higher-Order Logic* (Springer)
- Norrish, M. 2008, *A Formal Semantics for C++*, Tech. rep., NICTA
- Popek, G. J., Kampe, M., Kline, C. S., et al. 1979, *Managing Requirements Knowledge*, *International Workshop on*, 0, 355
- Rushby, J. M. 1981, *SIGOPS Oper. Syst. Rev.*, 15, 12