Research Proposal
NWO Open Competition 2008

1 Project data

Project title

Security by Logic for Multithreaded applications

Project Acronym

SlaLoM

Principal Investigator

Dr. Marieke Huisman

Renewed Application

This is a new application.

2 Summary

English Summary

This project develops a uniform verification framework for the protection of data. Key innovation on which the proposal is based is the notion of self-composition. This gives a different view on classical security properties, recasting them into safety properties of a single program, and allows reuse of existing program verification techniques. We believe that this approach can handle a wide range of data-related security properties, such as confidentiality, integrity and anonymity, in a uniform way, allowing easier comparison. To make the framework usable for realistic applications, which interact with their environment, we concentrate on multithreaded applications, and properties that specify complete executions of an application.

In earlier work, we have shown feasibility of the approach by translating the confidentiality problem for multithreaded programs into a model checking problem. However, to make the approach scale, we propose to address the following topics: (*a*) widening the scope of the studied security properties; (*b*) development of a realistic program model, containing the main features of a multithreaded language like Java, including unbounded dynamic thread creation and termination; (*c*) use of parametrised versions of temporal logic formalisms, to be able to express properties over infinite data domains; (*d*) development of decision procedures for appropriate probabilistic properties; (*e*) reuse and adaptation of existing tools for algorithmic verification; and (*f*) definition of appropriate abstractions, to allow verification of infinite state space programs.

Abstract for Laymen (in Dutch)

Dit project ontwikkelt een verificatiemethode voor gegevensbescherming, één van de voorwaarden voor het garanderen van de veiligheid van hedendaagse applicaties. De technische innovatie die ten grondslag ligt aan dit voorstel is het begrip *zelf-compositie*, d.w.z. de mogelijkheid om een programma met zichzelf samen te stellen. Dit geeft de mogelijkheid om veiligheid van een programma uit te drukken

als een eigenschap van één enkele programma-executie, terwijl traditionele methoden meerdere executies met elkaar vergelijken. Voor eigenschappen die slechts één programma-executie beschrijven kunnen we welbekende standaard programmaverificatietechnieken gebruiken om te laten zien dat het programma deze eigenschappen daadwerkelijk heeft. Het voordeel van deze aanpak is dat verschillende belangrijke veiligheidseigenschappen (bijvoorbeeld vertrouwelijkheid van gegevens, integriteit van gegevens en anonimiteit) allemaal met dezelfde algorithmische methoden aangetoond kunnen worden.

In eerder werk hebben we al laten zien dat deze aanpak mogelijk is: eigenschappen die de vertrouwelijkheid van gegevens beschrijven kunnen geverifieerd worden d.m.v. het doorlopen van de toestandsruimte van het programma (het zgn. *model checking*). In dit project breiden we deze eerste resultaten uit, om de techniek geschikt te maken voor een grotere klasse van eigenschappen en programma's. Hiervoor zullen we krachtigere specificatieformalismen en verificatietechnieken ontwikkelen, met daarbij natuurlijk de noodzakelijke software-ondersteuning. Twee essentiële uitbreidingen daarbij zijn het toevoegen van onbegrensde data aan de programmamodellen en eigenschappen en het gebruik van kansberekening om de veiligheidseigenschappen uit te drukken.

3 Classification

Discipline: Computer Science

The most relevant themes from NOAG-ict 2005-2010 (the Dutch national research agenda for computer science) are:

- Digital Security
  - Privacy
  - Protection
- Methods for Design and Construction
  - Software Engineering
  - Specification
  - Verification
  - Security

4 Composition of the Research Team

Principal Investigator Dr. Marieke Huisman is an expert on Java program verification and an active contributor to the Java Modeling Language (JML), a specification language for Java. She is currently involved in the development of BML, the bytecode variation of JML [8, 11]. She has been one of the main players in the development of the LOOP verification tool [22, 30, 28, 27, 24], has proposed static analyses and extensions for JML [39, 9], and has worked on formal methods for smart cards [33, 7]. Recently, she has also worked on model checking information flow properties [29, 26] and on proof preserving parallellisation. Further, she is involved in a long running project on the compositional verification of control flow properties for programs with procedures [20, 19, 25], and in the development of a separation-logic-based proof system for resource properties of multithreaded programs [21].

Huisman participates in the European FET Integrated Project Mobius (see `http://mobius.inria.fr`), where she leads the task on the verification of multithreaded applications, and is a member of the Management Committee of the ESF

(European Science Foundation) COST action on "Formal Verification of Object-Oriented Software" (since November 2007). COST provides funding for network activities, e.g., workshops and extended research visits, not staff.

Huisman obtained her PhD from the University of Nijmegen in the Netherlands. She worked 8 years at INRIA (as post doc and as *chargée de recherche* - researcher), and recently (August 1, 2008) she took up a special tenure track position for women at the University of Twente. She is appointed as assistant professor in the Formal Methods and Tools group (led by Prof. dr. Jaco van de Pol), and, based on quality of her work, will become an associate professor within five years.

Other team members Other members of the Formal Methods and Tools group are also involved in the project, to advise on issues related with their particular expertise. This includes in particular Prof. dr. Jaco van de Pol, Dr. Arend Rensink, associate professor, and Dr. Mariëlle Stoelinga, assistant professor on a tenure track position, to become an associate professor within 5 years. Van de Pol is an expert in process-algebra-based modeling, testing and model checking. Rensink is an expert in the application of graph transformations for specification and verification of dynamic behaviour, and for verification of design-time models and model transformation. In addition, he has worked on model checking object-oriented programs. Finally, Stoelinga is an expert in the analysis (in particular using formal verification, testing and game theory) of quantitative and extra-functional properties of embedded systems, and in particular of probabilistic, hybrid, and timed systems.

Within the project, a PhD student will be appointed. At the moment, there is no particular candidate for this position. Appropriate announcements will be made via mailing-lists and contacts of team members. Huisman will be daily supervisor of the PhD student, while Van de Pol is planned promotor.

name	involvement (fte)	specialism	university
Dr. M. Huisman	0.2	program verification, specification languages	U. of Twente
Prof. dr. J.C. van de Pol	0.1	process algebra, model checking	U. of Twente
Dr. A. Rensink	p.m.	graph transformations	U. of Twente
Dr. M. Stoelinga	p.m.	probabilistic, hybrid, and timed systems, quantitative analysis	U. of Twente
project PhD student	1.0		U. of Twente

5 Research School

The research will be embedded in the University of Twente research institute CTIT. In addition, the research group at the University of Twente is part of the national research school *Institute for Programming Research and Algorithmics* (IPA, www.win.tue.nl/ipa). The proposed research falls within IPA's themes *Formal Methods* and *Software Technology and Engineering*. Its relevant IPA focus area is *Software Analysis*.

6 Description of the Proposed Research

This project develops a uniform verification framework for the protection of data. Key innovation on which the proposal is based is the notion of self-composition. This provides a different view on classical security properties, recasting them into safety

properties of a single program, and allows to reuse existing program verification techniques for efficient and precise verification. We believe that this approach can handle a wide range of data-related security properties in a uniform way, allowing for easier comparison. Moreover, because of the use of general program verification techniques, the approach can easily be extended to newly proposed security properties.

Motivation and Challenges There cannot be much doubt that there is a growing need for formal means to guarantee security of applications, as for example illustrated by the recent media debate around the hacking of the *OV-chip card* and the *Oyster card* (the public transport card in the Netherlands and London, respectively). If users have the feeling that they cannot safely use such a card (without the risk of for example people stealing their bank information, observing their travel behaviour, or modifying their travel history), they will refuse to use it - as witnessed by the withdrawal of the consumer organisations from the OV-chip card discussion platform. To avoid such situations, one needs clear procedures to develop such applications, *and* appropriate tools to give formal security assurances.

A complicating factor is that because of increased computing power, software is becoming more and more complex. In particular the use of multithreading complicates the traceability of an application, because of the possible interferences between different threads. Thus a manual code inspection is insufficient to guarantee security; instead the use of dedicated tools is required.

In addition, the term "security" covers a large range of properties, and for each of these, dedicated techniques and tools are developed. As mentioned above, within this project, we focus on the protection of data. Typical security properties that will be covered are confidentiality (no private information can be derived from public data), integrity (private data cannot be affected by public data), and anonymity (identity is private information that cannot be derived from other actions of the application). Typical for these properties is that they can be expressed over pairs of executions. For example, confidentiality is ensured if from any two executions starting with the same public data, but possibly different private data, no differences in the public data can be observed.

In the literature, different properties are proposed to ensure data protection [37]. However, classical definitions often restrict only the input-output behaviour of an application, which is often not the most appropriate model for realistic (security-sensitive) applications, since these interact and exchange intermediate results. Thus to show that notwithstanding these interactions and communications, data is properly protected, also reachable intermediate states need to be confined by the security properties. The literature contains several proposals for this (again focusing in particular on the confidentiality of data, see [36] for an overview), but they often impose severe restrictions on the accepted programs. Moreover, they often are not compositional, which means that security of individual applications does not imply security of their composition. And finally, these properties (and techniques to ensure these properties) are often expressed over idealised programming languages, that for example do not consider dynamic thread creation and termination.

Thus to summarise, two important challenges to guarantee data protection are (1) to develop verification techniques that also work for realistic multithreaded applications, including dynamic thread creation and termination, and (2) to express the appropriate security properties, that apply to software that interacts with its environment.

Approach In recent years, a promising approach to guarantee security has been the adaption of techniques from programming languages, see [37] for a survey. Work

in this area has in particular focused on the development of special type systems to ensure the protection of data. However, such a type-based approach is typically incomplete: since it is insensitive to control-flow, many secure applications will be rejected. Moreover, for multithreaded applications often serious restrictions are imposed on how applications can be written, so that they are amenable to security type checking.

A recent, exciting alternative proposes to use logic-based program verification techniques to verify security properties that can be expressed as properties over two executions [3, 15]. The key idea behind this approach is to compose an application with itself, in such a way that the original two copies still can be distinguished, and to rephrase the security requirements as properties over a single execution of this self-composed application. This allows one to reuse standard program verification techniques, thus giving a sound and potentially complete verification technique.

Within this project, the self-composition approach will be extended to multithreaded applications. Therefore, an appropriate model for multithreaded applications will be developed, and composed with itself. The security properties will be re-expressed as safety properties over this self-composed model. As explained above, the security properties will have to confine intermediate reachable states, therefore temporal logic or process equivalences will be used. This gives the possibility to reuse model checking, bisimulation or equivalence checkers and other algorithmic verification techniques for the verification of security properties.

First steps taken We have already done some work to extend the self-composition approach to multithreaded applications. As a first step, we rephrased observational determinism [42] (a property proposed to protect data of multithreaded applications) as a formula in temporal logic [29], and experimented with the CADP model checker [16]. Unfortunately, the alternation-free modal μ-calculus supported by CADP turned out not to be expressive enough for our purpose. Besides the rephrasing in temporal logic, our investigations also revealed that observational determinism on the one hand was a very restrictive property, while on the other hand it would accept programs containing a security breach.

Therefore, as a next step, we investigated other confidentiality properties for multithreaded applications that we could find in the literature (reported in [4, 26]). We rephrased these properties over a uniform program model, to allow easier comparison between the different proposals. We had expected to find at least some partial hierarchy between the different properties, but in fact they turned out all to be incomparable. Further, we found that we could divide the proposed security properties into two categories: those that required visible executions (*i.e.*, the public traces) to be deterministic, and those that required an equal probabilistic distribution over the visible executions. In both categories, we found that there were properties that came close to characterising our intuitive notion of confidentiality. Further, as a proof of concept, we rephrased the "deterministic" confidentiality properties into temporal logic, again using the idea of self-composition, and we used the Concurrency WorkBench (CWB) [12], which supports modal μ-calculus with alternations, to model check these over some simple examples. This succeeded for some properties and some very simple examples, limiting the state space to only a few values, but clearly more work is required to apply this to realistic examples.

Research Goals As mentioned above, final goal of this project is the development of a uniform verification framework for the protection of data, using the self-composition approach to re-express appropriate security properties as temporal logic formulae or process equivalences. This allows the reuse of appropriate algorithmic verification techniques (if necessary, after adaptation) for the verification

of security properties.

The earlier results described above show that the self-composition approach indeed makes precise algorithmic verification of security properties feasible. However, at the same time they also reveal many open problems that need to be solved before the technique can be used efficiently on realistic applications. This leads to the following list of concrete research topics that have to be addressed to achieve the final goals of the project:

a) widening the scope of the security properties studied;

b) development of a more realistic program model, containing the main features of a multithreaded language like Java, including unbounded dynamic thread creation and termination;

c) use of parametrised versions of temporal logic formalisms, to be able to express properties over infinite data domains;

d) development of decision procedures for appropriate probabilistic properties;

e) reuse and adaptation of existing tools for algorithmic verification (including tools supporting probabilities) to make verification scale to realistic programs;

f) definition of appropriate abstractions, to allow verification of infinite state space programs.

For each of these points, we briefly describe the relevant issues and possible solutions.

a) Scope of the security properties. So far, self-composition has been applied on confidentiality properties (or more precisely, on the technical *non-interference* property, from which confidentiality follows). However, many other security properties can be expressed as a property over two (or more) executions, which makes them suited for our technique. In particular, integrity of data can be ensured by the following technical property:

> in any two executions starting with the same private data, but possibly different public data, the private data should remain the same, *i.e.*, the public data does not affect changes to the private data.

We will also study characterisations of other security properties, such as anonymity (see *e.g.* [17]), authentication (see *e.g.* [13]) and availability (see *e.g.* [14]), to see if and how these can be expressed as properties over multiple executions. Also for these properties, many different proposals exist. Because of our approach, we can handle the general case where properties restrict the allowed traces of multithreaded executions.

As mentioned above, the properties in the literature for confidentiality are all incomparable, and moreover, none of them exactly matches our intuitive understanding of the property. Thus, we expect that studying the literature in itself is not sufficient, and we will also propose improved properties that match better with the intended notion of data protection.

b) Program model. Typically, verification techniques for security properties are developed for simplified programming languages. On the one hand, this is good, because many details of full programming languages are irrelevant to the security properties at hand. On the other hand, sometimes these simplifications are too restrictive, and severely limit their applicability. In particular, for multithreaded

applications, typically a fixed number of parallel threads is considered, while applications written in widely-used multithreaded programming languages as Java rely on the possibility to dynamically create and terminate threads (and thus have an unbounded number of parallel threads). These dynamic aspects have to be properly defined in the program model, so that Java programs can be directly mapped onto it. The basics for the program model will be a labelled transition system, derived directly from the operational semantics (as we did in earlier work [29, 4]), using our experience with control flow graph extraction from Java applications [25]. The focus on Java programs is important, since Java is currently the most-used language to develop security-critical applications.

c) Parametrised temporal logic. The key ingredient of the self-composition approach is that a security property over program P is rephrased into a safety property over program P, composed with itself (where each copy maintains an independent memory). Since we are interested in specifying allowed traces, temporal logic is a suitable formalism. In the earlier work described above, we used propositional modal μ-calculus [31] to characterise confidentiality properties. We found that we needed the expressiveness of alternating μ-calculus to be able to express that an execution in a program copy could be mimicked (*i.e.* repeated *w.r.t.* the publicly visible variables) by the other program copy.

However, many security properties are inherently data-dependent. For every possible value of a variable, a change to any possible other value has to be considered. In earlier work, we solved this problem by explicitly limiting the data domain. However, this is too restrictive, and moreover for any realistic application, this will still result in very large formulae. Therefore, we will investigate whether first-order μ-calculus [32], which includes data parameters and quantifiers, can be used. Recently *parametrised boolean equation systems* (PBES) [18] have been proposed as a technique for model checking this logic and to characterise equivalence of (possibly infinite) processes with data [10]. PBES have already been used to show anonymity over protocols with n participants [17], so they seem adequate to for security properties. However, in the specific case of this project, properties are verified w.r.t. program code. Therefore, we will investigate whether the use of PBES is appropriate, whether existing solving techniques for PBES [18, 10] can be used for the properties at hand, and if necessary, we will develop these further.

d) Verification of probabilistic properties. However, a generalisation of μ-calculus with parameters alone will not be sufficient to handle all security properties of interest, since confidentiality properties are often expressed using probabilities [40, 38]. To verify such properties, our basic approach will be to map programs to a probabilistic model, and then to apply probabilistic verification techniques to establish observational equivalence between the original program and its copy. Our target program model will be based on probabilistic modules [1], an extension of reactive modules [2] that support probabilistic assignments to variables and parallelism.

To prove confidentiality, we will in particular focus on partial probabilistic bisimulation [38] and variations thereof [4], a probabilistic notion of low bisimulation. This property has the advantage that it closely matches the intuitive understanding of secure information flow, and that it is compositional. We plan to devise efficient decision procedures for it, using that a program and its copy are independent, non-synchronising processes. In addition, we will study whether partial probabilistic bisimulations can also be used to express for example integrity and anonymity.

To mechanise the verification, we will provide an automatic translation to probabilistic modules, which is the input format of the probabilistic model checkers

Prism [23] and MRMC [41] and implement our probabilistic low-bisimulation procedure in MRMC, an open-source model checker developed in Aachen and Twente. Currently, MRMC does not support data, therefore we will also closely follow what happens within the STOP project (a FMT project), which aims at the combination of data and probabilities.

e) Tool support. To make the self-composition approach applicable for realistic Java programs, appropriate tool support is necessary. As mentioned above, a program model will be defined that captures the essential ingredients of multithreaded Java programs, in particular the unbounded number of dynamically created threads. This will be supported by a tool that extracts program models from Java applications, based on our experience with extraction of control flow graphs for Java applications [25], via the SOOT framework.

To provide tool support for verification, we will study existing tools, in particular those handling probabilistic and parametrised temporal logics (several of which are developed within the FMT group, *i.e.*, the parity game solver for full μ-calculus [34] and MRMC [41], or in groups with which close contacts exist), and to decide which one comes closest to implementing the algorithms that we need. Then, we will tailor the existing algorithms to our specific needs. We expect that we might have to make (small) changes to the core verification algorithms, but that this pragmatic approach will allow us to reuse a large part of the underlying infrastructure of such tools.

f) Abstractions. Initial experiments with the use of CWB to model check confidentiality properties showed that algorithmic verification is feasible, but time-consuming. Moreover, because of the focus on data, the models that we verify have infinite state spaces. For the initial experiments, we significantly reduced the state spaces, only considering a few possible values for each variable. To generalise to realistic programs, we need to use appropriate abstraction techniques. Several abstractions come to mind directly. One possibility is to abstract each variable to its level in the security lattice. However, this abstraction will often be too coarse. Another possibility is to distinguish states where we know that two variables (in the two program copies) are equal, and states where we know that the two variables are different - without actually caring what the actual values are. Finally, since we are dealing with applications where an unbounded number of threads can be created dynamically, we will also look into appropriate abstractions that handle this situation. In particular, several useful under- and over-approximations have been proposed in the literature (see *e.g.*, [35, 6, 5]). Experiments will be needed to decide how appropriate these abstractions are, and if more fine-tuning is necessary.

Embedding of the Proposed Research into the Current Research of the Group Our proposed research fits perfectly in the theme of the FMT group, that develops formal techniques and tools as a means to support the development of software, including probabilistic verification. Our verification approach based on self-composition complements other software verification and validation techniques being developed in the FMT group, but also clearly connects with ongoing research in which the group is involved, such as the STOP project, the NWO-focus projects MOQS (modeling of quantitative aspects) and VeriGem (tool support for PBES), and the FP7 project Quasimodo (quantitative system properties). This connection is also indicated by the involvement of several team members in the project.

Other Research Relations On the national level, several possibilities for collaboration exist. Since the project also has a clear security aspect, it will be a good

opportunity to collaborate with the DIES group, at the University of Twente. Further, we expect to collaborate with the Universities of Nijmegen (Digital Security group) and Eindhoven (the Formal Methods, Systems Engineering, and Security groups). Additionally, we expect collaboration within the context of the VEMPS project (on multi-party protocol security).

On the international level, Huisman plays an important role in the European Integrated Project Mobius, which still runs until October 2009. We will keep collaborating with other Mobius participants. The proposed research will also benefit from the collaboration and contacts within the ESF COST action on "Formal Verification of Object-Oriented Software".

Finally, on a bilateral basis, we expect to collaborate with Dr. T. Rezk, INRIA Sophia Antipolis, France, Dr. G. Barthe, IMDEA, Spain, and Dr. P. D'Argenio, FaMAF, Córdoba, Argentina, all experts in formal methods for security and original proposers of the self-composition approach.

Application Perspective As explained above, formal techniques to guarantee security of applications are important for the wide-spread acceptance of new techniques such as the OV-chip card. Within the project, we do not plan to develop an industrially usable tool, but we do aim for a prototype that can be used to demonstrate feasibility of the approach in a convincing way. Because of the focus on algorithmic verification techniques, and the goal to make verification as precise as possible, we believe that the results produced by the project will be an important step forward towards industrially usable software to guarantee various security aspects of security-critical software.

7 Project Planning

Year 1: Literature study of security properties, uniform formalisation of security properties based on self-composition using temporal logic, process equivalence and probabilities; comparison and improvement of characterisations. Expected result: collection of formal security properties, expressed in (an extension of) temporal logic, expressing classical properties like confidentiality, integrity and anonymity, suitable for multithreaded applications, and matching intuitive understanding of "security" as close as possible.

Year 2: Formalisation of appropriate program model, capturing aspects such as dynamic thread creation and termination, implementation of a tool that can map Java programs to program models, appropriate (adaptation of) verification techniques for (parametrised) temporal logic, process equivalences and probabilistic bisimulations used to capture security properties. Expected result: algorithmic support for security properties of Java programs, using as much as possible existing tools and techniques. Since no specific tailoring to the security properties is developed yet, verification might be inefficient and not scale to infinite state spaces.

Year 3: Development and implementation of appropriate abstraction techniques, tailored to security properties at hand. Expected result: adaptation of existing tools and techniques, allowing efficient verification of security properties for (possibly infinite state) multithreaded applications.

Year 4: Application of techniques to several case studies, and improvements of developed tools and techniques where necessary. Writing of thesis.

The PhD student will participate in training activities offered by the research school IPA, and is expected to attend relevant summer schools (e.g., Marktoberdorf or LASER).

8 Expected Use of Instrumentarium

None, except standard desktop equipment. FMT has a verification cluster for large computation jobs.

9 Literature

Relevant references

Five relevant references of the applicants are:

1. M. Huisman, P. Worah, and K. Sunesen. A temporal logic characterisation of observational determinism. In *19th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, July 2006.

2. D. Gurov, M. Huisman, and C. Sprenger. Compositional verification of sequential programs with procedures. *Information and Computation*, 206:840–868, 2008.

3. M. Huisman, I. Aktug, and D. Gurov. Program models for compositional verification. In *International Conference on Formal Engineering Methods (ICFEM '08)*, volume 5256 of *LNCS*, pages 147–166. Springer Verlag, 2008.

4. T. Chen, S.C.W. Ploeger, J.C. van de Pol, and T.A.C Willemse. Equivalence checking for infinite systems using parameterized boolean equation systems. In *Concurrency Theory (CONCUR 2007)*, volume 4703 of *Lecture Notes in Computer Science*, pages 120–135. Springer, 2007.

5. C. Haack, M. Huisman, and C. Hurlin. Reasoning about Java's reentrant locks. In *Asian Programming Languages and Systems Symposium*, December 2008. To appear.
   Available from http://www-sop.inria.fr/everest/Clement.Hurlin/publis/sljrl.pdf.

References

[1] L. de Alfaro, T.A. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In *Concurrency Theory (CONCUR)*, volume 2154 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2001.

[2] R. Alur and T.A. Henzinger. Reactive modules. *Formal Methods in System Design*, 15:7–48, 1999.

[3] G. Barthe, P. D'Argenio, and T. Rezk. Secure Information Flow by Self-Composition. In R. Foccardi, editor, *Proceedings of CSFW'04*, pages 100–114, Pacific Grove, USA, June 2004. IEEE Press.

[4] H.-C. Blondeel. Security by logic: characterizing non-interference in temporal logic. Master's thesis, KTH Sweden, 2007. Available from `ftp://ftp-sop.inria.fr/everest/Marieke.Huisman/blondeel.pdf`.

[5] A. Bouajjani, J. Esparza, S. Schwoon, and J. Strejček. Reachability analysis of multithreaded software with asynchronous communication. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS '05)*, volume 3821 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2005.

[6] A. Bouajjani, J. Esparza, and T. Touili. A generic approach to the static analysis of concurrent programs with procedures. *SIGPLAN Notes*, 38(1):62–73, 2003.

[7] C. Breunesse, N. Cataño, M. Huisman, and B. Jacobs. Formal methods for smart cards: an experience report. *Science of Computer Programming*, 55(1-3):53–80, 2005.

[8] L. Burdy, M. Huisman, and M. Pavlova. Preliminary design of BML: A behavioral interface specification language for Java bytecode. In *Fundamental Approaches to Software Engineering (FASE 2007)*, volume 4422 of *Lecture Notes in Computer Science*, pages 215–229. Springer-Verlag, 2007.

[9] N. Cataño and M. Huisman. Chase: a static checker for JML's assignable clause. In L.D. Zuck, P.C. Attie, A. Cortesi, and S. Mukhopadhyay, editors, *Verification, Model Checking and Abstract Interpretation*, volume 2575 of *Lecture Notes in Computer Science*, pages 26–40. Springer, 2003.

[10] T. Chen, S.C.W. Ploeger, J.C. van de Pol, and T.A.C Willemse. Equivalence checking for infinite systems using parameterized boolean equation systems. In *Concurrency Theory (CONCUR 2007)*, volume 4703 of *Lecture Notes in Computer Science*, pages 120–135. Springer, 2007.

[11] J. Chrzaszcz, M. Huisman, A. Schubert, J. Kiniry, M. Pavlova, and E. Poll. *BML Reference Manual*, 2008. In progress. Available from http://www.jmlspecs.org/BML.

[12] R. Cleaveland, J. Parrow, and B. Steffen. The concurrency workbench: A semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15(1):36–72, January 1993.

[13] C.J.F. Cremers, S. Mauw, and E.P. de Vink. Injective synchronisation: An extension of the authentication hierarchy. *Theoretical Computer Science*, 367:139–161, 2006.

[14] F. Cuppens, N. Cuppens-Boulahia, and T. Ramard. Availability enforcement by obligations and aspects identification. In *Availability, Reliability and Security (AReS 2006)*, 2006.

[15] A. Darvas, R. Hähnle, and D. Sands. A theorem proving approach to analysis of secure information flow. In R. Gorrieri, editor, *Workshop on Issues in the Theory of Security*. IFIP WG 1.7, ACM SIGPLAN and GI FoMSESS, 2003.

[16] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2006: A toolbox for the construction and analysis of distributed processes. In *19th International Conference on Computer Aided Verification (CAV 2007)*, volume 4590 of *Lecture Notes in Computer Science*, pages 158–163. Springer, 2007.

[17] J.F. Groote and S. Orzan. Parameterised anonymity. In *Proceedings FAST'08*, Lecture Notes in Computer Science. Springer, 2008. To appear.

[18] J.F. Groote and T.A.C. Willemse. Parameterised boolean equation systems. *Theoretical Computer Science*, 343:332–369, 2005.

[19] D. Gurov and M. Huisman. Reducing behavioural to structural properties of programs with procedures. In *Verification, Model Checking, and Abstract Interpretation (VMCAI 2009)*, LNCS. Springer, 2009. To appear.

[20] D. Gurov, M. Huisman, and C. Sprenger. Compositional verification of sequential programs with procedures. *Information and Computation*, 206:840–868, 2008.

[21] C. Haack, M. Huisman, and C. Hurlin. Reasoning about Java's reentrant locks. In *Asian Programming Languages and Systems Symposium*, December 2008. To appear. Available from http://www-sop.inria.fr/everest/Clement.Hurlin/publis/sljrl.pdf.

[22] U. Hensel, M. Huisman, B. Jacobs, and H. Tews. Reasoning about classes in object-oriented languages: Logical models and tools. In C. Hankin, editor, *Proceedings of European Symposium on Programming (ESOP '98)*, volume 1381 of *Lecture Notes in Computer Science*, pages 105–121. Springer, 1998.

[23] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *Lecture Notes in Computer Science*, pages 441–444. Springer, 2006.

[24] M. Huisman. *Reasoning about Java Programs in Higher Order Logic with PVS and Isabelle*. PhD thesis, University of Nijmegen, 2001.

[25] M. Huisman, I. Aktug, and D. Gurov. Program models for compositional verification. In *International Conference on Formal Engineering Methods (ICFEM '08)*, volume 5256 of *LNCS*, pages 147–166. Springer, 2008.

[26] M. Huisman and H.-C. Blondeel. Secure information flow for multi-threaded programs, 2008. Manuscript.

[27] M. Huisman and B. Jacobs. Inheritance in higher order logic: Modeling and reasoning. In J. Harrison and M. Aagaard, editors, *Theorem Proving in Higher Order Logics: 13th International Conference (TPHOLs 2000)*, volume 1869 of *Lecture Notes in Computer Science*, pages 301–319. Springer, 2000.

[28] M. Huisman and B. Jacobs. Java program verification via a Hoare logic with abrupt termination. In T. Maibaum, editor, *Fundamental Approaches to Software Engineering (FASE 2000)*, volume 1783 of *Lecture Notes in Computer Science*, pages 284–303. Springer, 2000.

[29] M. Huisman, P. Worah, and K. Sunesen. A temporal logic characterisation of observational determinism. In *19th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, July 2006.

[30] B. Jacobs, J. van den Berg, M. Huisman, M. van Berkum, U. Hensel, and H. Tews. Reasoning about Java classes (preliminary report). In *Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'98)*, pages 329–340. ACM Press, 1998.

[31] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[32] D. Park. Finiteness is mu-ineffable. *Theoretical Computer Science*, 3(2):173–181, 1976.

[33] M. Pavlova, G. Barthe, L. Burdy, M. Huisman, and J.-L. Lanet. Enforcing high level security properties for applets. In J.-J. Quisquater, P. Paradinas, Y. Deswarte, and A.A. El Kalam, editors, *Cardis'04*, pages 1–16. Kluwer, 2004.

[34] J.C. van de Pol and M. Weber. A multi-core solver for parity games. In I. Černa and G. Lüttgen, editors, *Proceedings of PDMC 2008*, ENTCS, 2008. To appear.

[35] S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS '05)*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

[36] A. Russo and A. Sabelfeld. Securing Interaction between Threads and the Scheduler. In *Computer Security Foundations Workshop*. IEEE Press, 2006.

[37] A. Sabelfeld and A. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communication*, 21:5–19, January 2003.

[38] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *Computer Security Foundations Workshop*, pages 200–215, Cambridge, UK, July 2000. IEEE Press.

[39] K. Trentelman and M. Huisman. Extending JML Specifications with Temporal Logic. In H. Kirchner and C. Ringeissen, editors, *Algebraic Methodology And Software Technology (AMAST'02)*, volume 2422 of *Lecture Notes in Computer Science*, pages 334–348. Springer, 2002.

[40] D. Volpano and G. Smith. Probabilistic noninterference in a concurrent language. In *Computer Security Foundations Workshop*, pages 34–43, Rockport, Massachusetts, June 1998. IEEE Press.

[41] I. S. Zapreev. *Model Checking Markov Chains: Techniques and Tools*. PhD thesis, University of Twente, Enschede, The Netherlands, 2008.

[42] S. Zdancewic and A.C. Myers. Observational determinism for concurrent program security. In *16th IEEE Computer Security Foundations Workshop*, 2003.

10 Requested Budget

The PhD student that will be hired within the project is expected to collaborate with one of the project's international collaborators during a long-term visit (1 or 2 months, most likely with Dr. T. Rezk at INRIA Sophia Antipolis). In addition, we request some money to finance visits from our collaborators to the University of Twente.

appointment PhD student	(standard amount)	177,495 eur
bench fee	(standard amount)	5,000 eur
long visit to INRIA Sophia Antipolis	(estimate)	3,000 eur
2 visits from international collaborators	(estimate)	1,500 eur
total		186,995 eur