Research Proposal
NWO Open Competition 2014

1 Project data

1a Project title

Verification of Distributed Software

1b Project Acronym

VerDi

1c Application Module

2

1d Details of the Applicant

Title: Dr.
First name: Marieke
Last name: Huisman
Initials: M.
Female

Address for correspondence:
University of Twente
Formal Methods and Tools
Building Zilverling
Postbus 217
7500 AE Enschede

Preference for correspondence in English: no
Telephone: +31 53 489 46 62
Cell phone: +31 6 45 85 27 07
Fax: +31 53 489 32 47
Email: M.Huisman@utwente.nl
Website: http://wwwhome.ewi.utwente.nl/∼marieke/

1e Extension Clause

I obtained my PhD in 2001. My children are born in 2005 and 2008, which gives me the right to obtain a three years extension. Additionally, I have taken special caring leave to take care of my son when he was ill in the period between March and August 2008. This has been communicated with Mark van Assem from NWO, who confirmed (on March 13, 2014) that I am still eligible to apply for Module 2.

1f Scientific Summary

The VerDi project studies the Verification of Distributed Data Structures. Its aim is to develop automated program-logic based techniques for (message-based) distributed software.

Developing techniques to ensure the correctness of such programs is of utmost importance, because they are widely used for many different applications, including safety-critical ones. Typical examples are air traffic control systems and communication networks for emergency services. At the same time, this is also a considerable challenge, because in distributed programs the exchanged messages can arrive in arbitrary order or even get lost, so that a single program can show many different behaviours. Current research on the verification of distributed programs focuses mostly on message exchange between the different sites, modelled at an abstract level, without considering implementation details. The VerDi project will bridge the gap between the abstract and the implementation level, and enable reasoning about the actual distributed software.

The VerDi project builds on my experience and ongoing work in the VerCors project, where a specific program logic, called separation logic, is used to verify concurrent software i.e., programs that run on a single computer. Within the VerDi project, I aim to show that separation logic is also suitable to verify distributed programs running on multiple computers. In particular, it will be used to explore distributed implementations of data structures. To specify the behaviour of data structures, it is essential that the logic is really on the level of the program code. All techniques will be validated on realistic and relevant examples.

1g Keywords

Distributed software, Multithreading, Program logics, Separation logic, Verification

1h Abstract for Laymen (in Dutch)

Recentelijk zijn er in de media regelmatig berichten verschenen over problemen die veroorzaakt worden door softwarefouten. Doordat ICT steeds meer geïntegreerd is in ons dagelijks leven, hebben deze fouten ook steeds grotere consequenties. Het doel van het VerDi project is om een belangrijk gedeelte van deze fouten te voorkomen.

Het opsporen van fouten in software is lastig. Software wordt steeds groter en complexer, waardoor een software-ontwikkelaar of -evalueerder makkelijk het overzicht verliest. Daarnaast is software ook niet meer sequentieel: in moderne software zijn er meerdere executiepaden die naast elkaar uitgevoerd worden, maar wel hetzelfde geheugen moeten delen. Sterker nog, veel applicaties draaien verspreid over een netwerk, waarbij elke site in het netwerk moet communiceren met de andere sites. Typische voorbeelden van dit soort *gedistribueerde systemen* zijn communicatiesystemen voor hulpdiensten (zoals het C2000 systeem) en vlucht-controlesystemen. Voor veel van deze systemen is het essentieel dat ze in alle gevallen correct functioneren (en dat is helaas lang niet altijd het geval). Omdat communicatie tussen verschillende sites lang niet altijd betrouwbaar is – berichten kunnen elkaar inhalen, of zelfs helemaal niet aankomen – is het ondoenlijk om simpelweg systematisch alle mogelijke gedragingen van zo'n programma te analyseren. In plaats daarvan moeten er formele redeneertechnieken gebruikt worden om aan te tonen dat er nooit iets verkeerd zal gaan, en deze technieken moeten door krachtige softwaretools ondersteund worden.

Er bestaat veel onderzoek om het gedrag van gedistribueerde systemen op hoog niveau te beschrijven. Hiervoor zijn verschillende theoretische modellen ontwikkeld. Om echter met 100 % zekerheid te kunnen garanderen dat een applicatie correct werkt, moet ook de daadwerkelijke implementatie gecontroleerd worden. Dat is precies wat het VerDi project beoogd. De techniek die binnen VerDi gebruikt zal worden om over software te redeneren is gebaseerd op het idee van een programmalogica. Dit is een klassieke methode om te redeneren over alle mogelijke gedragingen van een programma, daterend uit de jaren zestig. Gedurende de laatste 15 jaar is

deze methode geautomatiseerd en toegepast op moderne programmeertalen, zoals Java. Dit onderzoek heeft er toe geleid dat moderne software-ontwikkelomgeving allerlei veelvoorkomende fouten in software automatisch kunnen detecteren.

De laatste vijf jaar heeft dit onderzoek zich vooral geconcentreerd op het redeneren over programma's waarin meerdere executies naast elkaar plaats vinden – waarbij het geheugen gedeeld wordt. Dit heeft geleid tot de ontwikkeling van *permission-based separation logic*, een programmalogica waarmee problemen in het gebruik van het gedeelde geheugen gedetecteerd kunnen worden.

Binnen het VerDi project gaan we deze techniek verder aanpassen, zodat deze ook geschikt wordt voor gedistribueerde applicaties. Hiervoor zullen we ons in eerste instantie richten op de specificatie en verificatie van gedistribueerde datastructuren. Datastructuren beschrijven opslagmechanismes voor allerlei gegevens. Zij zijn een belangrijke bouwsteen voor allerlei applicaties en daarom is het essentieel dat zij correct functioneren. Vervolgens zullen de technieken ook toegepast worden op applicaties die deze datastructuren gebruiken. Alle ontwikkelde methoden zullen door tools ondersteund worden en geëvalueerd worden aan de hand van realistische en relevante voorbeelden.

1i Main field of research

Computer Science
16.30.00 Theoretical computer science
16.20.00: Software, algorithms, control systems

1j Relevance to the 'Top Sectors'

This project falls in the top sector High-Tech Systems and Materials (as also confirmed by the ICT Roadmap). It is related to the ICT Roadmap Action Line 'ICT one can rely on', and with the Challenge 'How do we create secure and robust ICT?', identified in the Scientific Answer to the ICT Roadmap.

2 Research Plan

2a Description of the Proposed Research

Overall Aim

Software is becoming more and more complex. Many applications are multithreaded and distributed, either because this is the natural solution to the problem at hand, or to meet the ever-increasing demands on performance. Many safety-critical applications, such as air traffic control and emergency services communication systems fall in this category. Because of the ever-increasing complexity, it is a major difficulty to correctly implement such applications.

The overall goal of the VerDi project is to develop techniques to reason about program code that is both distributed and multithreaded. VerDi will use the results of my on-going ERC Starting Grant project VerCors that studies the verification of concurrent programs, and adapt and extend these to a whole new application domain, namely that of distributed programs.

The applications that we are interested in consist of different threads of execution, distributed over different sites, communicating and collaborating to reach the overall goal of the software. Communication can happen for example via message passing or bulletin boards, but we focus primarily on message passing. Moreover,

communication may be faulty, *i.e.*, messages may be received out of order, or be lost[1]. Multiple threads of execution running in parallel on one site typically use a single shared global memory, whereas executions that run on different sites have their own local copy of (part of) the memory. These local memories may overlap, i.e., two sites each might have a local copy of the same variables. Messages are passed between the different sites to ensure that the local memories remain consistent, and that their combination gives a coherent view of the overall state of the application. This model leads to an exponentially large set of possible program behaviours, thus making it a major challenge to prove that the application indeed functions correctly under all circumstances.

At the same time, we see that critical tasks are increasingly left to software. Thus developing correct software is not only becoming more and more complex, but also more and more essential. Therefore, it is of the utmost importance to develop techniques to reason formally about the correctness of multithreaded and distributed software [20], and the VerDi project will do exactly this. It should be emphasised that VerDi focuses on program code and not on software models. At a higher level of abstraction, well-known techniques, such as process algebras [19, 27, 5] and π-calculus [28, 34], are used to reason about models of concurrent and distributed programs. There also exist some proposals to use separation logic to reason about a language with process algebraic communication operators [21, 3]. However, there is still a large gap between these approaches and verification of actual distributed program code, and VerDi will bridge this gap.

Theory on program verification of sequential programs dates back to the sixties and seventies [11, 18, 13]. Over the last two decades, this has been integrated into several powerful tools [4, 2, 10]. An important advantage of using program logics is that one can reason about unbounded data structures and infinite data domains. Moreover, program logic-based approaches have proven to be suitable to verify realistic case studies (*e.g.* [12]).

Over the last years, a new program logic, called *separation logic*, has been developed that explicitly reasons about the heap [33]. This logic provides at least a partial solution to the verification problem for multithreaded applications [29, 16, 14, 22, 26, 32]. Currently, within the context of my ERC Starting Grant project VerCors, I use it to verify functional correctness of general-purpose, widely-used programming languages, such as Java, supported by automated tool support [2].

Key Objectives

The expected outcome of the VerDi project will be a powerful framework to reason about *realistic distributed software*.

Concretely, the expected outcomes of VerDi are:

- a sound and complete program logic to reason about multithreaded and distributed software, with an accompanying tool set;
- examples of fully specified and verified reusable distributed classes, such as distributed data structures; and
- a case study that demonstrate feasibility of the approach on a realistic example.

The main verification techniques will be based on separation logic. Tool support will be integrated in the VerCors tool set [2], which currently supports reasoning about concurrent Java and OpenCL kernels. Advantage of this approach is that this allows one to reason about both concurrency and distribution aspects of an application in a single framework. The VerDi project will develop specifications

[1] However, reorderings still have to obey certain restrictions, which make verification feasible.

and verifications of widely-used, general-purpose classes to be used in a distributed context, such as distributed data structures. Correctness of such classes is important, as they can be reused in a wide range of different applications. The particular language of interest will be Java, but the theory will be developed generically, to make it also applicable to other programming languages.

Background

Already in the seventies, Owicki and Gries developed a logic to reason about multiple threads [30]. However, their approach was non-compositional, and exponential in the number of proof obligations. Later, Jones developed a compositional approach: rely-guarantee reasoning [24]. However, this was still too complex to be used on any realistic application. It was only in 2007 that O'Hearn showed how separation logic could be used to reason about multithreaded programs [29]. The main idea is that separation logic allows one to state explicitly which part of the program accesses which part of the memory. If two threads access disjoints parts of the memory, then they cannot interfere, and their correctness can be shown in a thread-modular way, see e.g., [14, 22, 26, 32].

Together with Haack and Hurlin [17, 16, 15, 23], I pioneered to apply this approach to Java, by extending separation logic with permissions [9, 8] (reusing separation logic for sequential Java, developed by Parkinson [31]). Permissions guarantee that multiple threads can simultaneously read a memory location, while at most one thread at the time can write a location. Soundness of the logic ensures that as long as a thread holds a read permission on a location, this location cannot be changed by any (other) thread, and thus the value stored at this location is stable. The logic has been applied on many non-trivial patterns of concurrent programming, see `https://fmt.ewi.utwente.nl/redmine/projects/vercors-verifier/wiki/Online_Examples` for various examples. Reasoning is supported by the VerCors tool set [2], which combines separation logic specifications with classical program specifications techniques (concretely the specification language JML, a behavioural interface specification language for Java [25]).

When reasoning about distributed software, it is well-understood how this can be modelled and validated at a high-level of abstraction, using for example process algebras. However, from the VerDi point of view, this is design-time verification, and there is still a large gap to the verification of the implementation. Some work exists on model checking distributed implementations, see e.g., [35], but this typically takes an abstract view of the code, considering only method calls. There also exist some program logics to reason about programs using process algebra operations for communication [3, 21]. However, if one wishes to reason in full detail about a distributed program, also considering the data, one needs to apply a combination of process algebras, model checking and program logic on concrete software, using APIs for the implementation of the distribution aspects. And, because of the parallelism that is inherent to distributed programming, a classical Floyd-Hoare logic is not sufficient: instead a permission-based separation logic, combined with JML, as developed by my team, is the appropriate choice. In this logic, permissions should not only manage the concurrency within a single site; instead the permission mechanism has to be extended to also regulate that data cannot be freely changed, because it is replicated and should remain consistent with data on other sites.

Originality and Success Factors

Even though the VerDi project will build on the VerCors project, it will *not* be just a continuation of it.

First of all, the VerDi project will open up a completely new application domain for program logics. More concretely, it will extend the use of separation logic and the permission model to distributed software. Second, it will combine program logic with specification and verification techniques from proces algebra and model checking, enabling program logics to be used in completely new ways. And finally, because of the unique combination of techniques to reason about concurrent *and* distributed programs, the result will be a powerful framework to reason about *realistic distributed code*.

Moreover, the techniques are also applicable for modern special-purpose hardware, with multiple processors that can access only parts of the memory. The VerDi project will lead to a whole new class of applications that can be completely verified, using a single specification language to describe and verify all relevant aspects of the implementation.

A critical success factor is that we already have the VerCors tool in which the results of VerDi can be integrated. Having tool support is an absolute requirement if one wishes to reason about non-trivial applications. Moreover, the VerDi theory can build on our theory for concurrent programs, in particular our approach to reason about class invariants [3] and variable evolution (in progress).

Research Plans

Goal of the VerDi project is to develop specification and verification techniques for distributed software. As a starting point, special attention will be given to Java programs using the MPJ API (for example, applications where high performance is essential); however, the logic will be developed in such a way that it can be adapted to other programming languages as well. Concretely, the following tasks will be addressed:

- *A program logic for distributed programs*: an extension of permission-based separation logic for distributed programs, to reason about data replication between sites.
- *Specification and verification of a collection of general-purpose, widely-used, reusable distributed classes*, to validate applicability of the logic.
- *Tool support*, applicable to real-world examples.
- *A case study*, to evaluate the whole approach.

Approach

Basis of the work is the permission-based separation logic that I developed with Haack and Hurlin [15, 17, 16, 23]. This logic allows one to reason about a Java-like language with reentrant locks, dynamic thread creation and termination in a thread-modular way. The logic contains the standard operators from separation logic, i.e., the points-to predicate, the separating conjunction (the star operator), and the separating implication (or magic wand). In addition, each pointer is associated with a permission that specifies whether the owner of the pointer can either read or update the location that is being pointed to. Permissions are a value in the domain (0, 1]. Soundness of the logic ensures a global correctness criterion, namely that the total number of permissions for a particular location never exceeds 1. If a thread has a pointer with permission 1, this means that this thread is the only thread that can access this location, and thus the location can safely be modified. If a thread only has a fractional permission, i.e., a permission that is less than 1, other threads might also be accessing the same location simultaneously, and thus the value stored there can only be read, and not updated. An essential feature of this approach is that a program can only be verified with this logic, if it does not contain any data races (and of course, if it respects the method specifications).

Below, I describe how the different tasks mentioned above will be addressed within the project.

A program logic for distributed software. The basis of the project is the extension of permission-based separation logic described above to reason about distributed data structures and applications, making it suitable to reason about code that is both distributed and concurrent.

A major challenge is that data can be distributed over different sites, thus not only concurrent accesses within the data have to be handled, but also the locations at which the data is stored, and the fact that the data may be replicated. To achieve this, the logic will be extended with rules to maintain consistency of data that is replicated on multiple sites in the network, e.g., by not allowing any changes to it, or by enforcing that any change is communicated to the other sites. A first step thus will be to investigate and define an appropriate notion of consistency. Further, the underlying heap model has to extend the standard heap model, to consider that data can be distributed over different sites, and might be replicated. Finally, the rules in the logic will be adapted and proven sound w.r.t. this extended distributed heap model.

We plan to start by using the MPJ API as the concrete message passing implementation. This means that we have to annotate the operations in this interface appropriately using our specification language. The separation logic specifications for process algebras by Bell *et al.* [3] and Hoare and O'Hearn [21] will form an important inspiration, and we will consider whether we can derive the MPJ specifications from these. In a later stage, we might also look at other message passing or remote method invocation APIs. Additionally, we will also consider whether the reference implementation of the API can be verified w.r.t. our specification.

Specification and verification of general-purpose distributed classes. The logic will be used to specify and verify implementations of several general-purpose distributed classes, which can be reused in many different applications, such as distributed hash tables, and vectors. Development of the logic will be done in parallel with writing these specifications, to ensure practical usability of the logic.

The specifications will express the integrity of the data, i.e., all data that is stored can eventually be retrieved, the data that is stored will not be changed without proper permissions, and if applicable, the data will be retrieved at the appropriate moment. In a sequential setting, these specifications are standard. However, in a concurrent and distributed setting, one has to ensure that other threads cannot invalidate the guarantees that are specified for an operation. For example, when specifying the add operation of a hash table, one cannot state that after the operation has terminated, the element is indeed stored in the hash table and is located on-site. In the meantime, before the caller of this add operation continues its execution, another thread might have already removed the data again, or moved it to another site. Instead, in this case a solution is to give specifications in terms of variable evolution; keeping track of the fact that there was a point in time where the element was stored on-site in the data structure.

Tool support. Within the VerDi project, the VerCors tool set will be extended to a distributed setting. Currently, the tool set supports the verification of multithreaded Java programs, and of vector programs written as OpenCL kernels, annotated with a combination of permission-based separation logic and JML. It is intended to be practically usable and to have a high level of automation, leveraging existing provers, such as Chalice [26] and Boogie [1], for the different verification tasks.

The VerDi tool set will be built on top of this existing tool set, taking advantage of its features to reason efficiently about multithreaded programs, and adding the means to reason about distributed software. The development of the tool set will be done in close collaboration with Stefan Blom, who is responsible for the VerCors tool set. The development will be done continuously throughout the project, as the logic and the specifications develop.

Case study. During the second half of the project, the techniques will be validated on a larger, real-world example. Possible sources of case studies are:

- a distributed implementation of some service for online shopping from Fredhopper/SDL (see also 2b on knowledge utilisation of the VerDi project), and
- distributed data structures and algorithms on labelled transition systems used in the LTSmin tool set [7] for distributed model checking.

Collaboration and Details of Research Group

The VerDi project will take place within the Formal Methods and Tools (FMT) group at the University of Twente. The FMT group is one of the most successful and productive computer science groups in the department. It is embedded in the CTIT research institute, and via my participation in the CTIT centre for dependable systems and networks, and the CTIT centre for cyber security and public safety, I also have contacts with members of other related groups at the university.

As explained above, within the group, synergy is expected with the VerCors team and the LTSmin tool set team. In addition, my international contacts will provide useful collaborations with other teams working on topics such as program verification (of concurrent applications), separation logic and distributed programming. Contacts exists for example with the Programming Methodology group at ETH Zürich, Switzerland; the Software Engineering using Formal Methods group, at Chalmers University, Sweden; the DistriNet group, at the University of Leuven, Belgium; the Precise Modelling and Analysis group, at the University of Oslo, Norway and other members of the HATS project; the Programming, Logic, and Semantics group at ITU Copenhagen, Denmark; and the Research in Software Engineering group at Microsoft Research.

To Conclude

The goal of the VerDi project is to develop specification and verification techniques for distributed software. Because of the widespread use of distributed programs, including for safety-critical applications, the results of the VerDi project will be highly relevant and have a huge potential for cost savings in code development and maintenance. Moreover, because of the expected synergy with the VerCors project, the VerDi project will lead to a single framework to verify concurrent and distributed implementations efficiently.

Key project objectives are:

1. A collection of fully specified and verified implementations of reusable, general-purpose distributed classes.
2. A sound verification technique for concurrent and distributed programs, where data is distributed over different sites, but has to respect global consistency conditions.
3. Tool support for all the developed techniques.
4. Evidence of feasibility of the approach by applying the techniques on a real-life example.

Project Planning

To execute the work plan detailed above, support is asked for one (4 year) PhD position. An initial planning for the project is given below, but of course, developments during the project might require a deviation from this work plan. Notice that during the 4th year of the project, time is explicitly reserved for writing up the thesis.

Year 1: *Program Logic for Distributed Software*: Specification of data moves, selecting appropriate notion of consistency
Distributed Classes: Specification of distributed hash table

Year 2: *Program Logic for Distributed software*: Heap model and soundness proof, improving the logic
Tool Support: Start implementation of Distributed Logic
Distributed Classes: Verification of hash table, specification of distributed vector

Year 3: *Program Logic for Distributed software*: Complete soundness proof, fine-tuning of the logic
Distributed Classes: Verification of vector, specification and verification of other reusable, general-purpose distributed classes
Tool Support: Continue implementation of Distributed Logic

Year 4: *Case Study*: Distributed Model Checking Algorithm, IBIS Component, or distributed service
Thesis write-up

2b Knowledge Utilisation

To ensure that within the VerDi project, we can have significant impact, it is important to consider the industrial needs for software verification from the beginning of the project. I have good contacts with several companies, such as SDL/Fredhopper, a service provider for online webshops, Collaborne, a service provider for enterprise collaboration platforms, and Philips Healthcare, that encounter problems in the development of distributed software in their daily software development process. It is my intention that the solutions that we develop within VerDi will be directly applicable for such companies.

To achieve this, I plan to have regular contacts with them during the whole duration of the project. In particular, at the beginning of the project, the PhD candidate and myself will visit the companies, and identify concrete examples of problems in the development of distributed software. In particular I would like to have a good understanding of what possible execution scenario's are problematic, and thus should be avoided.

These examples will serve as one of the guiding factors when developing the verification techniques. Throughout the project, we will keep contact and have regular meetings (at least once a year, but more frequently if appropriate) to discuss our progress. In particular, we we will demonstrate our solutions, and obtain feedback about what can be done to make the techniques more applicable in an industrial context.

At the end of the project, we would like to have a prototype version of the tools available online, which can be used by interested users. To create interest, we will present our approach at industry-oriented meetings, such as Bits & Chips' High-tech or Smart Systems, CeBIT, and ICT.Open.

2c Literature

References

[1] M. Barnett, B.-Y. E. Chang, R. DeLine, B. Jacobs, and K. R. M. Leino. Boogie: A modular reusable verifier for object-oriented programs. volume 4111 of *Lecture Notes in Computer Science*, pages 364 – 387, 2005.

[2] B. Beckert, R. Hähnle, and P.H. Schmitt, editors. *Verification of Object-Oriented Software: The KeY Approach*, volume 4334 of *Lecture Notes in Computer Science*. Springer, 2007.

[3] C.J. Bell, A.W. Appel, and D. Walker. Concurrent separation logic for pipelined parallelization. In *Static Analysis Symposium (SAS 2010)*, 2010.

[4] J. van den Berg and B. Jacobs. The LOOP compiler for Java and JML. In T. Margaria and W. Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, volume 2031 of *Lecture Notes in Computer Science*, pages 299–312. Springer, 2001.

[5] J.A. Bergstra, A. Ponse, and S.A. Smolka, editors. *Handbook of Process Algebra*. Elsevier Publishing, 2001.

[6] S. Blom and M. Huisman. The VerCors Tool for verification of concurrent programs. In C. Jones, P. Pihlajasaari, and J. Sun, editors, *Formal Methods (FM) 2014*, volume 8442 of *LNCS*, pages 127–131. Springer, 2014.

[7] S.C.C. Blom, J.C. van de Pol, and M. Weber. LTSmin: Distributed and symbolic reachability. In T. Touili, B. Cook, and P. Jackson, editors, *Computer Aided Verification (CAV 2010)*, volume 6174 of *Lecture Notes in Computer Science*, pages 354–359. Springer, 2010.

[8] R. Bornat, P. W. O'Hearn, C. Calcagno, and M. Parkinson. Permission accounting in separation logic. In J. Palsberg and M. Abadi, editors, *Principles of Programming Languages*, pages 259–270. ACM Press, 2005.

[9] J. Boyland. Checking interference with fractional permissions. In R. Cousot, editor, *Static Analysis Symposium*, volume 2694 of *Lecture Notes in Computer Science*, pages 55–72. Springer-Verlag, 2003.

[10] D. Cok and J. R. Kiniry. ESC/Java2: Uniting ESC/Java and JML: Progress and issues in building and using ESC/Java2 and a report on a case study involving the use of ESC/Java2 to verify portions of an internet voting tally system. In *Construction and Analysis of Safe, Secure and Interoperable Smart Devices (CASSIS 2004)*, volume 3362 of *Lecture Notes in Computer Science*, pages 108–128. Springer, 2005.

[11] E.W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[12] G. Ernst, G. Schellhorn, W. Reif, and D. Haneberg. The development of a fully verified flash file system, 2010.

[13] R. W. Floyd. *Assigning meanings to programs*. J. T. Schwartz editor, Mathematical Aspects of Computer Science. American Mathematical Society, 1967.

[14] A. Gotsman, J. Berdine, B. Cook, N. Rinetzky, and M. Sagiv. Local reasoning for storable locks and threads. In Z. Shao, editor, *Asian Programming Languages and Systems Symposium*, volume 4807 of *Lecture Notes in Computer Science*, pages 19–37. Springer-Verlag, 2007.

[15] C. Haack, M. Huisman, and C. Hurlin. Reasoning about Java's reentrant locks. In G. Ramalingam, editor, *Asian Programming Languages and Systems Symposium*, volume 5356 of *Lecture Notes in Computer Science*, pages 171–187. Springer-Verlag, December 2008.

[16] C. Haack, M. Huisman, and C. Hurlin. Permission-based separation logic for Java, 2011. Submitted.

[17] C. Haack and C. Hurlin. Separation logic contracts for a Java-like language with fork/join. In J. Meseguer and G. Rosu, editors, *Algebraic Methodology and Software Technology*, volume 5140 of *Lecture Notes in Computer Science*, pages 199–215. Springer-Verlag, July 2008.

[18] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[19] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21:666–677, 1978.

[20] C. A. R. Hoare and J. Misra. Verified software: Theories, tools, experiments vision of a grand challenge project. A companion paper for VSTTE 2005, held in Züich, Switzerland., July 2005.

[21] T. Hoare and P. O'Hearn. Separation logic semantics of communicating processes. In *1st FICS conference*, ENTCS, 2008.

[22] A. Hobor, A. Appel, and F.Zappa Nardelli. Oracle semantics for concurrent separation logic. In S. Drossopoulou, editor, *Programming Languages and Systems: Proceedings of the 17th European Symposium on Programming, ESOP 2008*, volume 4960 of *Lecture Notes in Computer Science*, pages 353–367. Springer-Verlag, 2008.

[23] C. Hurlin. Specifying and checking protocols of multithreaded classes. In *Symposium on Applied Computing*, pages 587–592. ACM Press, March 2009.

[24] C. B. Jones. Tentative steps toward a development method for interfering programs. *ACM Transactions on Programming Languages and Systems*, 5(4):596–619, 1983.

[25] G.T. Leavens, E. Poll, C. Clifton, Y. Cheon, C. Ruby, D.R. Cok, P. Müller, J. Kiniry, and P. Chalin. *JML Reference Manual*, February 2007. Department of Computer Science, Iowa State University. Available from http://www.jmlspecs.org.

[26] K.R.M. Leino, P. Müller, and J. Smans. Verification of concurrent programs with Chalice. In *Lecture notes of FOSAD*, volume 5705 of *Lecture Notes in Computer Science*. Springer, 2009.

[27] R. Milner. *A Calculus of Communicating Systems*. Springer, 1980.

[28] R. Milner. *Communicating and mobile systems - the Pi-calculus*. Cambridge University Press, 1999.

[29] P. W. O'Hearn. Resources, concurrency and local reasoning. *Theoretical Computer Science*, 375(1–3):271–307, 2007.

[30] S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica Journal*, 6:319–340, 1975.

[31] M. Parkinson. Local reasoning for Java. Technical Report UCAM-CL-TR-654, University of Cambridge, 2005.

[32] M.J. Parkinson, R. Bornat, and P.W. O'Hearn. Modular verification of a non-blocking stack. In M. Hofmann and M. Felleisen, editors, *Principles of Programming Languages (POPL 2007)*, pages 297–302. ACM, 2007.

[33] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Logic in Computer Science*, pages 55–74. IEEE Computer Society, 2002.

[34] Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.

[35] A. Vo, S. S. Vakkalanka, M. Delisi, G. Gopalakrishnan, R. M. Kirby, and R. Thakur. Formal verification of practical mpi programs. In Daniel A. Reed and Vivek Sarkar, editors, *PPOPP*, pages 261–270. ACM, 2009.

[36] Marina Zaharieva-Stojanovski and Marieke Huisman. Verifying class invariants in concurrent programs. In *Fundamental Approaches to Software Engineering*, volume 8411 of *LNCS*, pages 230–244. Springer, 2014.

2d Relevant references

Five relevant references of the applicants are:

1. S.C.C. Blom, M. Huisman, and M. Mihelčić. Specification and verification of GPGPU programs. Science of Computer Programming, to appear, 2014.

2. S.C.C. Blom, M. Huisman. The VerCors Tool Set for Verification of Concurrent Programs. In *Formal Methods 2014*, volume 8442 of *Lecture Notes in Computer Science*, pages 127-131, Springer-Verlag, 2014.

3. M. Zaharieva-Stojanovski, and M. Huisman. Verifying Class Invariants in Concurrent Programs. In *Fundamental Aspects of Software Engineering (FASE 2014)*, volume 8411 of *Lecture Notes in Computer Science*, pages 230-244, Springer-Verlag, 2014.

4. A. Amighi, S. Blom, M. Huisman, W. Mostowski, and M. Zaharieva-Stojanovski. Formal Specifications for Java's Synchronisation Classes. In *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2014)*, pages 725-733, IEEE Computer Society, 2014.

5. C. Haack, M. Huisman, and C. Hurlin. Reasoning about Java's reentrant locks. In G. Ramalingam, editor, *Asian Programming Languages and Systems Symposium*, volume 5356 of *Lecture Notes in Computer Science*, pages 171–187. Springer-Verlag, 2008.

3 Administrative Details

3a Budget

The PhD student that will be hired within the project is expected to spend 3 months in a related research group abroad (e.g., at ETH Zürich, University of Leuven, or at Microsoft Redmond) to broaden the scope of his research.

To cover the costs of such a visit, we request 5,000 euro more, leading to the following overall budget:

Appointment PhD student		(standard amount)
Long-term visit abroad	(estimate)	5,000 euro
Total		1 PhD student + 5,000 euro

3b Grant Applications

This proposal has been submitted before to NWO Free Competition in 2012, but it has not been funded. Compared to the earlier proposal, this proposal is more clear about the kind of distributed applications that will be considered.

3c Open Positions

Not applicable

4 Curriculum Vitae

Personal details
Dr. Marieke Huisman (female)
Born 3rd of May, 1973, in Utrecht, Netherlands
Nationality: Dutch

Master's
Computer Science
University of Utrecht, Netherlands
Graduation: 31/08/1996

Doctorate
Radboud University Nijmegen, Netherlands
Date: 01/02/2001
Java program verification in higher order logic with PVS and Isabelle
Supervisors: Prof.Dr. H.P. Barendregt, Dr. B.P.F. Jacobs, Dr.Ir. H. Meijer

Current employment and Work Experience since Completing the PhD

2011 -	Associate Professor, FMT Group, led by Prof.Dr. Jaco van de Pol, University of Twente (permanent, 0.8 fte)
2008 - 2011	Assistant Professor, FMT Group, University of Twente (tenure track, 0.8 fte)
2001 - 2008	Chargée de Recherche (researcher), first in the Lemme project, later in the Everest project, led by Dr. Gilles Barthe at INRIA Sophia Antipolis (permanent, initially 1 fte, since February 2006 0.8 fte)
2000 - 2001	Post doc INRIA Sophia Antipolis (fixed-term, 1 fte)

Note that even though my contract is permanent, I am still on a tenure track, meaning that agreements have been made between me and the UT about the necessary steps and timeline to become a full professor.

Man-years of research

Computation:

I.	1 year post doc at INRIA, 1.0 FTE	12 months
II.	Chargée de recherche	
	52 months 1.0 FTE,	
	minus maternity leave (Dec. 2004 Aug. 2005 = 9 months)	43 months
	30 months 0.8 FTE,	
	minus absence caused by illness son (4 months)	20.8 months
III.	Assistant professor	
	33 months 0.8 FTE,	
	minus maternity leave (4 months),	
	60 % of working time for research	13.9 months
IV.	Associate professor	
	34 months 0.8 FTE,	
	60 % of working time for research	16.3 months
	Total man-months of research	106 months
	Total man-years of research	8.8 years

Additional information:
At INRIA, I was appointed as a full-time researcher, without teaching obligations. At the University of Twente, my working time is partially spent on teaching and management.

For medical reasons, my first maternity leave was longer than standard. In 2008, my son was seriously ill, and this required the presence of one of the parents continuously. Altogether, I was not able to work for 4 months during this period.

Brief summary of research over the last five years I am a well-established researcher in the area of program verification. During the last five years, I worked in particular on verification and information-flow properties of concurrent software. Additionally, I also have expertise in verification of sequential Java programs, both at sourcecode and bytecode level, the design of the Java Modeling Language (JML), and compositional program verification. I am also active in devising competitions for deductive program verification.

My work on concurrent software verification is supported by my ERC Starting Grant VerCors, and the EU project CARP. I developed a permission-based separation logic for Java, allowing to reason about dynamic thread creation and termination, and about different synchronisation mechanisms. At the moment, I work together with Afshin Amighi and Stefan Blom on deriving specifications for atomic operations, to prove correctness of different synchronisers. Additionally, together with Marina Zaharieva, I look at how we can verify functional properties for concurrent programs. We developed a technique to verify class invariants in a concurrent setting, and we are working on a technique to keep track of possible changes to variables. With Saeed Darabi, I have shown how permission-based separation logic is also suitable to reason about vector programs. Finally, together with Wojciech Mostowski, I have developed a symbolic permission model, enabling easier automatic reasoning about permissions.

To algorithmically verify information-flow properties of concurrent programs, we have proposed an improved definition of confidentiality in a concurrent setting, and

adapted existing model checking algorithms to a security context for verification.

International activities

- Coordinator of an EU Horizon2020 project proposal DIAMOND (Differential Models for Continuous Deployment)
- Site leader for EU Strep project CARP (Correct and Efficient Accelerator Programming), 2011-2014.
- Site leader and Task leader of the task on Multithreading in the IP FET project Mobius (Mobility, Ubiquity and Security, 2005-2009). Coordinator of the Specific Task Force that defines BML (Bytecode Modeling Language). Contributed to submission and coordination of this project (see http://mobius.inria.fr).
- Coordinator of collaborative project with KTH, Sweden on compositional verification of control-flow security properties for applets (2005), funded by INRIA, France.
- 3 weeks visit to ANU Canberra, Australia, collaboration with Kerry Trentelman.
- 3 months visit to the Automated Reasoning Group, led by Mike Gordon and Larry Paulson, at Cambridge University, UK (1999).

Other academic activities

Invited Presentations

- Lecturer at SPES_XT Summer School at AVoCS'14 on Model-based design and analysis of cyber-physical systems, 2014.
- Lecturer at SFM-14:ESM - 14th International School on Formal Methods for the Design of Computer, Communication and Software Systems: Executable Software Models, 2014.
- Dutch Theory Day, NVTI, 2014.
- ICT.Open, 2013.
- A Celebration of Formal Methods, FME AGM, 2013.
- Annual meeting of RS3 DFG Priority Program, 2012.
- Platform Parallel Netherlands GPGPU-day, 2012.
- International Key Workshop, 2011.
- Dutch Model Checking Day, 2010.
- 3rd Dutch Proof Tools Day, 1998.

PhD and Evaluation Committees

- PhD Committee Violet Ka I Pun
  Behavioural Static Analysis for Deadlock Detection
  University of Oslo, Norway, 2014.
- Member Advisory Committee associate professorship Michel Reniers
  Technical University Eindhoven, Netherlands, 2013.
- Member Advisory Committee professorship Aiko Pras
  University of Twente, Netherlands, 2013.
- PhD Committee Hannes Mehnert
  Incremental Interactive Verification of the Correctness of Object-Oriented Software
  IT University, Denmark, 2013.
- PhD Committee Mattias Ulbrich
  Dynamic Logic for an Intermediate Language
  Karlsruhe Institute of Technology, Germany, 2013.

- PhD Committee Frédéric Vogels
  Formalisation and Soundness of Static Verification Algorithms for Imperative Programs
  University of Leuven, Belgium, 2012.
- Discussion leader for licentiate thesis Ran Ji
  Towards a Deductive Compilation Approach
  Chalmers University of Technology, Sweden, 2012.
- Member Advisory Committee professorship Frank Kargl
  University of Twente, Netherlands, 2012.
- External advisor for a Tenure Track evaluation, 2011.
- Manuscript committee Alejandro Tamalet
  Towards Correct Programs in Practice. Proving Functional and Non-functional Properties by means of Program Analysis
  Radboud University Nijmegen, Netherlands, 2011.
- Member Advisory Committee professorship Arend Rensink
  University of Twente, 2010.
- Committee member Joachim van den Berg
  Reasoning about Java programs in PVS using JML
  Radboud University Nijmegen, Netherlands, 2009.

Academic Leadership and Coordination

- Member of Selection Committee for NWO Free Competition (2010, 2011), VIDI Competition (2014), Leraren Competition (2014)
- Member of Program Board Informatics Lorentz center Leiden (since January 2012)
- Member of Management Committee ESF COST Action on Formal Verification of Object-Oriented Software.
- Site-coordinator for the French national research project Geccoo (2003 2006), see http:// geccoo.lri.fr).
- Coordination of the French national research project ModoCop (Model Checking of Concurrent Object Oriented Programs, 2001 2003).

Organizational and editorial duties

- Co-organiser VerifyThis program verification competitions at ETAPS 2015, FM 2012, FoVeOos 2011.
- Co-organiser AVOCS workshop and associated summer school, 2015.
- Workshop co-chair Formal Methods 2015.
- Co-proposer Lorentz seminar *JML: Advancing Specification Language Methodologies*
- Co-organiser Lorentz seminar *Reliability of Concurrent and Distributed Programs*
- Co-editor special issue of STTT for VerifyThis@FM2012.
- Co-organiser Dagstuhl seminar *Evaluating Software Verification Systems: Benchmarks and Competitions*, 2014.
- Editor special issue of SCP for Bytecode 2012.
- Co-organiser Dagstuhl seminar *Correct and Efficient Accelerator Programs*, April 2013.
- Co-Proposer Dagstuhl seminar *Divide and Conquer: the Quest for Compositional Design and Analysis*, December 2012.
- Editor special issue of JOT for FTfJP and IWACO 2008.
- Member of the jury for the Isabelle Attali best technical paper award, e-Smart 2005 and 2006.
- Editor special issue of JLAP on Formal Methods for Smart Cards, 2004.
- Organiser of final workshop of ModoCop project, Grenoble, 2003.

- Organiser of VeriSafe workshop (for VerifiCard and SecSafe projects), Nice, 2002.
- Organiser of Lemme project seminar, INRIA Sophia Antipolis, 2000 2001.
- Organiser of Computing Science Institute seminar, University of Nijmegen, Netherlands, 1998 2000.
- Organiser of ProTaGoNist (Proof Tools Group Netherlands), and 2nd Dutch Proof Tools Day, University of Nijmegen, Netherlands, 1997.

Program committees

- TCS 2014
- AVOCS 2014 (co-chair)
- Verify 2014 (IJCAR workshop)
- FOAL 2014 (Modularity workshop)
- PLDI 2014 (extended review committee)
- OOPSLA 2013 (extended review committee)
- integrated Formal Methods (iFM 2013)
- FMi 2013, 2014 (IEEE IRI workshop)
- Bytecode 2013, 2012 (chair), 2007 (co-chair), 2005 (ETAPS workshop)
- ESSS2013, 2013, 2014 (ICST and FM workshop)
- FTSCS 2012, 2013, 2014 (ICFEM workshop)
- Svarm-Verify 2012 (IJCAR workshop)
- Compare 2012 (IJCAR workshop)
- PLAS 2012 (PLDI workshop)
- VMCAI 2012
- Fossacs 2011
- FoVeOos 2010, 2011
- SAVCBS 2009 (chair), 2007, 2006 (ESEC/FSE workshop)
- FTfJP 2008 (chair), 2007, 2004 (ECOOP workshop)
- VAMP 2007 (Concur workshop)
- VSTTE 2006 (part of FloC)
- .NET Technologies 2006, 2004
- Cassis 2005 (co-chair), 2004 (co-chair)

Scholarships, grants and prizes

- Nominated for Computer Science Educational Award at University of Twente, 2014.
- Netherlands Prize for ICT Research 2013, a unique prize for a scientist, aged 40 years or younger for innovative research or a scientific breakthrough in ICT (50,000 euros).
- National role model EU Campaign: Science, its a girl thing!, 2012.
- EU Strep FP7 project CARP (Correct and Efficient Accelerator Programming, 2011 2014), funding for 1 PhD student (co-applicant).
- ERC Starting Grant VerCors (Verification of Concurrent Data Structures, 2011 2016), 1,3 ME, funding for 2 PhD students, 2 post docs and 0.2 FTE programmer during 5 years (PI).
- NWO Free Competition project SlaLom (Security by Logic of Multithreaded Applications, 2010-2014), funding for 1 PhD student during 4 years (PI).
- IP FET project Mobius (Mobility, Ubiquity and Security, 2005 2009), funding for 3 PhD students (co-applicant).
- EASST Best Paper Award at ETAPS 2004 for: M. Huisman, D. Gurov, C. Sprenger, and G. Chugunov. Checking absence of illicit applet interactions: a case study. In M. Wermelinger and T. Margaria, editors, Fundamental Approaches to Software Engineering (FASE 2004), number 2984 in LNCS,

pages 8498. Springer Verlag, 2004.
– French national research project ModoCop (Model Checking of Concurrent Object Oriented Programs, 2001 2003), funding for 3 months post doc (PI).
– Freye stipendium, travel grant for female PhD student to support long term visits abroad, used for a 3 months visit to the Automated Reasoning Group, led by Mike Gordon and Larry Paulson, at Cambridge University, UK (1999).

Ten top publications

1. S.C.C. Blom, M. Huisman, and M. Mihelčić. Specification and verification of GPGPU programs. Science of Computer Programming, to appear, 2014.
2. T.M. Ngo, M. Stoelinga, and M. Huisman. Effective Verification of Confidentiality for Multi-Threaded Programs. Journal of Computer Security 22, pages 269300, 2014.
3. M. Zaharieva-Stojanovski, and M. Huisman. Verifying Class Invariants in Concurrent Programs. In *Fundamental Aspects of Software Engineering (FASE 2014)*, volume 8411 of *Lecture Notes in Computer Science*, pages 230-244, Springer-Verlag, 2014.
4. A. Amighi, S. Blom, M. Huisman, W. Mostowski, and M. Zaharieva-Stojanovski. Formal Specifications for Java's Synchronisation Classes. In *Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2014)*, pages 725-733, IEEE Computer Society, 2014.
5. D. Gurov and M. Huisman. Reducing Behavioural to Structural Properties of Programs with Procedures. Theoretical Computer Science, 2013.
6. S. Soleimanifard, D. Gurov and M. Huisman. Procedure-Modular Specification and Verification of Temporal Safety Properties. *Software and Systems Modeling*, 2013.
7. S. Blom, J.R. Kiniry and M. Huisman. How Do Developers Use APIs? A Case Study in Concurrency. In International Conference on Engineering of Complex Computer Systems (ICECCS 2013). IEEE Computer Society, 2013.
8. A. Amighi, P. de Carvalho Gomes, D. Gurov and M. Huisman. Sound Control-Flow Graph Extraction for Java programs with Exceptions. In *SEFM 2012*. LNCS 7504, pages 33 - 47, Springer, 2012.
9. M. Huisman and A. Tamalet. A Formal Connection between Security Automata and JML Annotations. In *FASE 2009*. LNCS 5503, pp. 340-354, Springer, 2009.
10. D. Gurov, M. Huisman, and C. Sprenger. Compositional verification of sequential programs with procedures. *Information and Computation*, 206:840-868, 2008.