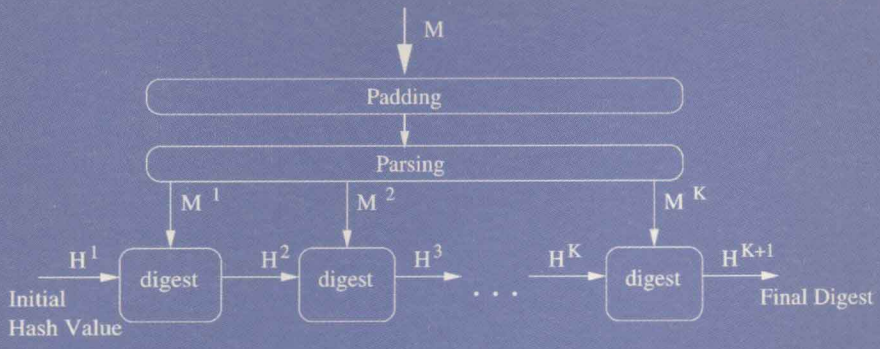
Hot Topics

LNCS 3362

Gilles Barthe
Lilian Burdy
Marieke Huisman
Jean-Louis Lanet
Traian Muntean (Eds.)

Construction and Analysis of Safe, Secure, and Interoperable Smart Devices

International Workshop, CASSIS 2004
Marseille, France, March 2004
Revised Selected Papers



Springer

Gilles Barthe Lilian Burdy
Marieke Huisman Jean-Louis Lanet
Traian Muntean (Eds.)

Construction and Analysis of Safe, Secure, and Interoperable Smart Devices

International Workshop, CASSIS 2004
Marseille, France, March 10-14, 2004
Revised Selected Papers

Springer

Volume Editors

Gilles Barthe
Lilian Burdy
Marieke Huisman
Jean-Louis Lanet
INRIA Sophia-Antipolis
2004 Route des Lucioles, BP 93, 06902 Sophia Antipolis, France
E-mail: {Gilles.Barthe, Marieke.Huisman, Jean-Louis.Lanet}@inria.fr
Lilian.Burdy@sophia.inria.fr

Traian Muntean
Université de la Méditerranée
Ecole Supérieure D'Ingénieurs de Luminy
Case 925 - ESIL Parc Scientifique, 13288 Marseille, France
E-mail: Traian.Muntean@esil.univ-mrs.fr

Library of Congress Control Number: 2004117384

CR Subject Classification (1998): D.2, C.3, D.1, D.3, D.4, F.3, E.3

ISSN 0302-9743
ISBN 3-540-24287-2 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Olgun Computergrafik
Printed on acid-free paper SPIN: 11375197 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3362

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Editorial Board

Lecture Notes in Computer Science

For information about Vols. 1–3260

please contact your bookseller or Springer

Vol. 3385: R. Cousot (Ed.), Verification, Model Checking, and Abstract Interpretation. XII, 483 pages. 2004.

Vol. 3382: J. Odell, P. Giorgini, J.P. Müller (Eds.), Agent-Oriented Software Engineering V. X, 239 pages. 2004.

Vol. 3381: M. Bieliková, B. Charon-Bost, O. Sýkora, P. Vojtáš (Eds.), SOFSEM 2005: Theory and Practice of Computer Science. XV, 428 pages. 2004.

Vol. 3363: T. Eiter, L. Libkin (Eds.), Database Theory - ICDT 2005. XI, 413 pages. 2004.

Vol. 3362: G. Barthe, L. Burdy, M. Huisman, J.-L. Lanet, T. Muntean (Eds.), Construction and Analysis of Safe, Secure, and Interoperable Smart Devices. IX, 257 pages. 2004.

Vol. 3360: S. Spaccapietra, E. Bertino, S. Jajodia, R. King, D. McLeod, M.E. Orlowska, L. Strous (Eds.), Journal on Data Semantics II. XI, 233 pages. 2004.

Vol. 3358: J. Cao, L.T. Yang, M. Guo, F. Lau (Eds.), Parallel and Distributed Processing and Applications. XXIV, 1058 pages. 2004.

Vol. 3356: G. Das, V.P. Gulati (Eds.), Intelligent Information Technology. XII, 428 pages. 2004.

Vol. 3353: J. Hromkovič, M. Nagl, B. Westfechtel (Eds.), Graph-Theoretic Concepts in Computer Science. XI, 404 pages. 2004.

Vol. 3350: M. Hermenegildo, D. Cabeza (Eds.), Practical Aspects of Declarative Languages. VIII, 269 pages. 2004.

Vol. 3348: A. Canteaut, K. Viswanathan (Eds.), Progress in Cryptology - INDOCRYPT 2004. XIV, 431 pages. 2004.

Vol. 3347: R.K. Ghosh, H. Mohanty (Eds.), Distributed Computing and Internet Technology. XX, 472 pages. 2004.

Vol. 3344: J. Malenfant, B.M. Østvold (Eds.), Object-Oriented Technology. ECOOP 2004 Workshop Reader. VIII, 215 pages. 2004.

Vol. 3342: E. Şahin, W.M. Spears (Eds.), Swarm Robotics. X, 175 pages. 2004.

Vol. 3341: R. Fleischer, G. Trippen (Eds.), Algorithms and Computation. XVII, 935 pages. 2004.

Vol. 3340: C.S. Calude, E. Calude, M.J. Dinneen (Eds.), Developments in Language Theory. XI, 431 pages. 2004.

Vol. 3339: G.I. Webb, X. Yu (Eds.), AI 2004: Advances in Artificial Intelligence. XXII, 1272 pages. 2004. (Subseries LNAI).

Vol. 3338: S.Z. Li, J. Lai, T. Tan, G. Feng, Y. Wang (Eds.), Advances in Biometric Person Authentication. XVIII, 699 pages. 2004.

Vol. 3337: J.M. Barreiro, F. Martin-Sanchez, V. Maojo, F. Sanz (Eds.), Biological and Medical Data Analysis. XI, 508 pages. 2004.

Vol. 3336: D. Karagiannis, U. Reimer (Eds.), Practical Aspects of Knowledge Management. X, 523 pages. 2004. (Subseries LNAI).

Vol. 3334: Z. Chen, H. Chen, Q. Miao, Y. Fu, E. Fox, E.-p. Lim (Eds.), Digital Libraries: International Collaboration and Cross-Fertilization. XX, 690 pages. 2004.

Vol. 3333: K. Aizawa, Y. Nakamura, S. Satoh (Eds.), Advances in Multimedia Information Processing - PCM 2004, Part III. XXXV, 785 pages. 2004.

Vol. 3332: K. Aizawa, Y. Nakamura, S. Satoh (Eds.), Advances in Multimedia Information Processing - PCM 2004, Part II. XXXVI, 1051 pages. 2004.

Vol. 3331: K. Aizawa, Y. Nakamura, S. Satoh (Eds.), Advances in Multimedia Information Processing - PCM 2004, Part I. XXXVI, 667 pages. 2004.

Vol. 3329: P.J. Lee (Ed.), Advances in Cryptology - ASIACRYPT 2004. XVI, 546 pages. 2004.

Vol. 3328: K. Lodaya, M. Mahajan (Eds.), FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science. XVI, 532 pages. 2004.

Vol. 3326: A. Sen, N. Das, S.K. Das, B.P. Sinha (Eds.), Distributed Computing - IWDC 2004. XIX, 546 pages. 2004.

Vol. 3323: G. Antoniou, H. Boley (Eds.), Rules and Rule Markup Languages for the Semantic Web. X, 215 pages. 2004.

Vol. 3322: R. Klette, J. Žunić (Eds.), Combinatorial Image Analysis. XII, 760 pages. 2004.

Vol. 3321: M.J. Maher (Ed.), Advances in Computer Science - ASIAN 2004. XII, 510 pages. 2004.

Vol. 3320: K.-M. Liew, H. Shen, S. See, W. Cai (Eds.), Parallel and Distributed Computing: Applications and Technologies. XXIV, 891 pages. 2004.

Vol. 3316: N.R. Pal, N.K. Kasabov, R.K. Mudi, S. Pal, S.K. Parui (Eds.), Neural Information Processing. XXX, 1368 pages. 2004.

Vol. 3315: C. Lemaître, C.A. Reyes, J.A. González (Eds.), Advances in Artificial Intelligence – IBERAMIA 2004. XX, 987 pages. 2004. (Subseries LNAI).

Vol. 3314: J. Zhang, J.-H. He, Y. Fu (Eds.), Computational and Information Science. XXIV, 1259 pages. 2004.

Vol. 3312: A.J. Hu, A.K. Martin (Eds.), Formal Methods in Computer-Aided Design. XI, 445 pages. 2004.

Vol. 3311: V. Roca, F. Rousseau (Eds.), Interactive Multimedia and Next Generation Networks. XIII, 287 pages. 2004.

Vol. 3309: C.-H. Chi, K.-Y. Lam (Eds.), Content Computing. XII, 510 pages. 2004.

Vol. 3308: J. Davies, W. Schulte, M. Barnett (Eds.), Formal Methods and Software Engineering. XIII, 500 pages. 2004.

Vol. 3307: C. Bussler, S.-k. Hong, W. Jun, R. Kaschek, D.. Kinshuk, S. Krishnaswamy, S.W. Loke, D. Oberle, D. Richards, A. Sharma, Y. Sure, B. Thalheim (Eds.), Web Information Systems – WISE 2004 Workshops. XV, 277 pages. 2004.

Vol. 3306: X. Zhou, S. Su, M.P. Papazoglou, M.E. Orlowska, K.G. Jeffery (Eds.), Web Information Systems – WISE 2004. XVII, 745 pages. 2004.

Vol. 3305: P.M.A. Sloot, B. Chopard, A.G. Hoekstra (Eds.), Cellular Automata. XV, 883 pages. 2004.

Vol. 3303: J.A. López, E. Benfenati, W. Dubitzky (Eds.), Knowledge Exploration in Life Science Informatics. X, 249 pages. 2004. (Subseries LNAI).

Vol. 3302: W.-N. Chin (Ed.), Programming Languages and Systems. XIII, 453 pages. 2004.

Vol. 3300: L. Bertossi, A. Hunter, T. Schaub (Eds.), Inconsistency Tolerance. VII, 295 pages. 2004.

Vol. 3299: F. Wang (Ed.), Automated Technology for Verification and Analysis. XII, 506 pages. 2004.

Vol. 3298: S.A. McIlraith, D. Plexousakis, F. van Harmelen (Eds.), The Semantic Web – ISWC 2004. XXI, 841 pages. 2004.

Vol. 3296: L. Bougé, V.K. Prasanna (Eds.), High Performance Computing - HiPC 2004. XXV, 530 pages. 2004.

Vol. 3295: P. Markopoulos, B. Eggen, E. Aarts, J.L. Crowley (Eds.), Ambient Intelligence. XIII, 388 pages. 2004.

Vol. 3294: C.N. Dean, R.T. Boute (Eds.), Teaching Formal Methods. X, 249 pages. 2004.

Vol. 3293: C.-H. Chi, M. van Steen, C. Wills (Eds.), Web Content Caching and Distribution. IX, 283 pages. 2004.

Vol. 3292: R. Meersman, Z. Tari, A. Corsaro (Eds.), On the Move to Meaningful Internet Systems 2004: OTM 2004 Workshops. XXIII, 885 pages. 2004.

Vol. 3291: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, Part II. XXV, 824 pages. 2004.

Vol. 3290: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE, Part I. XXV, 823 pages. 2004.

Vol. 3289: S. Wang, K. Tanaka, S. Zhou, T.W. Ling, J. Guan, D. Yang, F. Grandi, E. Mangina, I.-Y. Song, H.C. Mayr (Eds.), Conceptual Modeling for Advanced Application Domains. XXII, 692 pages. 2004.

Vol. 3288: P. Atzeni, W. Chu, H. Lu, S. Zhou, T.W. Ling (Eds.), Conceptual Modeling – ER 2004. XXI, 869 pages. 2004.

Vol. 3287: A. Sanfeliu, J.F. Martínez Trinidad, J.A. Carrasco Ochoa (Eds.), Progress in Pattern Recognition, Image Analysis and Applications. XVII, 703 pages. 2004.

Vol. 3286: G. Karsai, E. Visser (Eds.), Generative Programming and Component Engineering. XIII, 491 pages. 2004.

Vol. 3285: S. Manandhar, J. Austin, U.B. Desai, Y. Oyanagi, A. Talukder (Eds.), Applied Computing. XII, 334 pages. 2004.

Vol. 3284: A. Karmouch, L. Korba, E.R.M. Madeira (Eds.), Mobility Aware Technologies and Applications. XII, 382 pages. 2004.

Vol. 3283: F.A. Aagesen, C. Anutariya, V. Wuwongse (Eds.), Intelligence in Communication Systems. XIII, 327 pages. 2004.

Vol. 3282: V. Guruswami, List Decoding of Error-Correcting Codes. XIX, 350 pages. 2004.

Vol. 3281: T. Dingsøyr (Ed.), Software Process Improvement. X, 207 pages. 2004.

Vol. 3280: C. Aykanat, T. Dayar, İ. Körpeoğlu (Eds.), Computer and Information Sciences - ISCIS 2004. XVIII, 1009 pages. 2004.

Vol. 3279: G.M. Voelker, S. Shenker (Eds.), Peer-to-Peer Systems III. XI, 300 pages. 2004.

Vol. 3278: A. Sahai, F. Wu (Eds.), Utility Computing. XI, 272 pages. 2004.

Vol. 3275: P. Perner (Ed.), Advances in Data Mining. VIII, 173 pages. 2004. (Subseries LNAI).

Vol. 3274: R. Guerraoui (Ed.), Distributed Computing. XIII, 465 pages. 2004.

Vol. 3273: T. Baar, A. Strohmeier, A. Moreira, S.J. Mellor (Eds.), <<UML>> 2004 - The Unified Modelling Language. XIII, 454 pages. 2004.

Vol. 3272: L. Baresi, S. Dustdar, H. Gall, M. Matera (Eds.), Ubiquitous Mobile Information and Collaboration Systems. VIII, 197 pages. 2004.

Vol. 3271: J. Vicente, D. Hutchison (Eds.), Management of Multimedia Networks and Services. XIII, 335 pages. 2004.

Vol. 3270: M. Jeckle, R. Kowalczyk, P. Braun (Eds.), Grid Services Engineering and Management. X, 165 pages. 2004.

Vol. 3269: J. Lopez, S. Qing, E. Okamoto (Eds.), Information and Communications Security. XI, 564 pages. 2004.

Vol. 3268: W. Lindner, M. Mesiti, C. Türker, Y. Tzitzikas, A. Vakali (Eds.), Current Trends in Database Technology - EDBT 2004 Workshops. XVIII, 608 pages. 2004.

Vol. 3267: C. Priami, P. Quaglia (Eds.), Global Computing. VIII, 377 pages. 2004.

Vol. 3266: J. Solé-Pareta, M. Smirnov, P.V. Mieghem, J. Domingo-Pascual, E. Monteiro, P. Reichl, B. Stiller, R.J. Gibbens (Eds.), Quality of Service in the Emerging Networking Panorama. XVI, 390 pages. 2004.

Vol. 3265: R.E. Frederking, K.B. Taylor (Eds.), Machine Translation: From Real Users to Research. XI, 392 pages. 2004. (Subseries LNAI).

Vol. 3264: G. Paliouras, Y. Sakakibara (Eds.), Grammatical Inference: Algorithms and Applications. XI, 291 pages. 2004. (Subseries LNAI).

Vol. 3263: M. Weske, P. Liggesmeyer (Eds.), Object-Oriented and Internet-Based Technologies. XII, 239 pages. 2004.

Vol. 3262: M.M. Freire, P. Chemouil, P. Lorenz, A. Gravey (Eds.), Universal Multiservice Networks. XIII, 556 pages. 2004.

Vol. 3261: T. Yakhno (Ed.), Advances in Information Systems. XIV, 617 pages. 2004.

Preface

This volume contains a selection of refereed papers from participants of the workshop "Construction and Analysis of Safe, Secure and Interoperable Smart Devices" (CASSIS), held from the 10th to the 13th March 2004 in Marseille, France:

`http://www-sop.inria.fr/everest/events/cassis04/`

The workshop was organized by INRIA (Institut National de Recherche en Informatique et en Automatique), France and the University de la Méditerranée, Marseille, France. The workshop was attended by nearly 100 participants, who were invited for their contributions to relevant areas of computer science.

The aim of the workshop was to bring together experts from the smart devices industry and academic researchers, with a view to stimulate research on formal methods and security, and to encourage the smart device industry to adopt innovative solutions drawn from academic research.

The next generation of smart devices holds the promise of providing the required infrastructure for the secure provision of multiple and personalized services. In order to deliver their promise, the smart device technology must however pursue the radical evolution that was initiated with the adoption of multi-application smartcards. Typical needs include:

- The possibility for smart devices to feature extensible computational infrastructures that may be enhanced to support increasingly complex applications that may be installed post-issuance, and may require operating system functionalities that were not pre-installed. Such additional flexibility must however not compromise security.
- The possibility for smart devices to achieve a better integration with larger computer systems, through improved connectivity, genericity, as well as interoperability.
- The possibility for smart devices to protect themselves and the applications they host from hostile applications, by subjecting incoming applications to analyses that bring strong guarantees in terms of confidentiality or resource control.
- The possibility for application developers to establish through formal verification based on logical methods the correctness of their applications. In addition, application developers should be offered the means to convey to end-users or some trusted third party some verifiable evidence of the correctness of their applications.
- The possibility for smart devices to be modeled and proved correct formally, in order to achieve security evaluations such as Common Criteria at the highest levels.

In order to address the different issues raised by the evolution of smart devices, the workshop consisted of seven sessions featuring one keynote speaker and three or four invited speakers:

1. Trends in smart card research
2. Operating systems and virtual machine technologies
3. Secure platforms
4. Security
5. Application validation
6. Verification
7. Formal modeling

The keynote speakers for this edition were: Eric Vétillard (Trusted Logic), Ksheerabdhi Krishna (Axalto), Xavier Leroy (INRIA), Pieter Hartel (U. of Twente), K. Rustan M. Leino (Microsoft Research), Jan Tretmans (U. of Nijmegen), and J. Strother Moore (U. of Texas at Austin).

In addition, a panel chaired by Pierre Paradinas (CNAM), and further consisting of Jean-Claude Huot (Oberthur Card Systems), Gilles Kahn (INRIA), Ksheerabdhi Krishna (Axalto), Erik Poll (U. of Nijmegen), Jean-Jacques Quisquater (U. of Louvain), and Alain Sigaud (Gemplus), examined the opportunities and difficulties in adapting open source software for smart devices execution platforms.

We wish to thank the speakers and participants who made the workshop such a stimulating event, and the reviewers for their thorough evaluations of submissions. Furthermore, we gratefully acknowledge financial support from Conseil Général des Bouches-du-Rhône, Axalto, France Télécom R&D, Gemplus International, Microsoft Research and Oberthur Card Systems.

November 2004

Gilles Barthe
Lilian Burdy
Marieke Huisman
Jean-Louis Lanet
Traian Muntean

Organizing Committee

Gilles Barthe	INRIA Sophia Antipolis, France
Lilian Burdy	INRIA Sophia Antipolis, France
Marieke Huisman	INRIA Sophia Antipolis, France
Jean-Louis Lanet	INRIA DirDRI, France
Traian Muntean	University de la Méditerranée, Marseille, France

Reviewers

Cuihtlauac Alvarado
John Boyland
Michael Butler
Koen Claessen
Alessandro Coglio
Adriana Compagnoni
Pierre Crégut
Jean-Michel Douin
Hubert Garavel
Nikolaos Georgantas
Mike Gordon
Chris Hankin
Rene Rydhof Hansen
Klaus Havelund
Lex Heerink
Ludovic Henrio
Charuwalee Huadmai
Thierry Jéron
Rajeev Joshi
Florian Kammüller
Laurent Lagosanto
Yassine Lakhnech
Xavier Leroy
Gerald Lüttgen
Anil Madhavapeddy
Claude Marché
Ricardo Medel
Greg Morisett
Laurent Mounier
Christophe Muller
Alan Mycroft
Brian Nielsen
David von Oheimb
Arnd Poetzsch-Hefftner
Erik Poll
Christophe Rippert
Judi Romijn
Vlad Rusu
Peter Ryan
David Sands
Gerardo Schneider
Ulrik Pagh Schultz
David Scott
Robert de Simone
Christian Skalka
Oscar Slotosch
Kim Sunesen
Sabrina Tarento
Hendrik Tews
Mark Utting
Eric Vétillard
Willem Visser
Olivier Zendra
Elena Zucca

Table of Contents

Mobile Resource Guarantees for Smart Devices*

David Aspinall[1], Stephen Gilmore[1], Martin Hofmann[2],
Donald Sannella[1], and Ian Stark[1]

[1] Laboratory for Foundations of Computer Science, School of Informatics,
The University of Edinburgh
[2] Lehr- und Forschungseinheit für Theoretische Informatik, Institut für Informatik,
Ludwig-Maximilians-Universität München

Abstract. We present the Mobile Resource Guarantees framework: a system for ensuring that downloaded programs are free from run-time violations of resource bounds. Certificates are attached to code in the form of efficiently checkable proofs of resource bounds; in contrast to cryptographic certificates of code origin, these are independent of trust networks. A novel programming language with resource constraints encoded in function types is used to streamline the generation of proofs of resource usage.

1 Introduction

The ability to move code and other active content smoothly between execution sites is a key element of current and future computing platforms. However, it presents huge security challenges – aggravating existing security problems and presenting altogether new ones – which hamper the exploitation of its true potential. Mobile Java applets on the Internet are one obvious example, where developers must choose between sandboxed applets and working within a crippled programming model; or signed applets which undermine portability because of the vast range of access permissions which can be granted or denied at any of the download sites. Another example is open smart cards with multiple applications that can be loaded and updated after the card is issued, where there is currently insufficient confidence in available security measures to take full advantage of the possibilities this provides.

A promising approach to security is *proof-carrying code* [26], whereby mobile code is equipped with independently verifiable certificates describing its security properties, for example type safety or freedom from array-bound overruns. These certificates are condensed and formalised mathematical proofs which are by their very nature self-evident and unforgeable. Arbitrarily complex methods may be used by the *code producer* to construct these certificates, but their verification by the *code consumer* will always be a simple computation. One may compare this to the difference between the difficulty of producing solutions to combinatorial

* This research was supported by the MRG project (IST-2001-33149) which is funded by the EC under the FET proactive initiative on Global Computing.

G. Barthe et al. (Eds.): CASSIS 2004, LNCS 3362, pp. 1–26, 2005.

problems such as Rubik's cube or satisfiability, and the ease of verifying whether an alleged solution is correct or not.

A major advantage of this approach is that it sidesteps the difficult issue of *trust*: there is no need to trust either the code producer, or a centralized certification authority. If some code comes with a proof that it does not violate a certain security property, and the proof can be verified, then it does not matter whether the code (and/or proof) was written by a Microsoft Certified Professional or a monkey with a typewriter: the property is guaranteed to hold. The user does need to trust certain elements of the infrastructure: the code that checks the proof (although a paranoid user could in principle supply a proof checker himself); the soundness of the logical system in which the proof is expressed; and, of course, the correctness of the implementation of the virtual machine that runs the code – however these components are fixed and so can be checked once and for all. In any case, trust in the integrity of a person or organization is not a reliable basis for trusting that the code they produce contains no undiscovered accidental security bugs! In practice it seems best to take advantage of *both* existing trust infrastructures, which provide a degree of confidence that downloaded code is not malicious and provides desired functionality, *and* the strong guarantees of certain key properties provided by proof-carrying code.

Control of resources (space, time, etc.) is not always recognized as a security concern but in the context of smart cards and other small devices, where computational power and especially memory are very limited, it is a central issue. Scenarios of application which hint at the security implications include the following:

- a provider of distributed computational power may only be willing to offer this service upon receiving dependable guarantees about the required resource consumption;
- third-party software updates for mobile phones, household appliances, or car electronics should come with a guarantee not to set system parameters beyond manufacturer-specified safe limits;
- requiring certificates of specified resource consumption will also help to prevent mobile agents from performing denial of service attacks using bona fide host environments as a portal;

and the one of most relevance in the present context:

- a user of a handheld device, wearable computer, or smart card might want to know that a downloaded application will definitely run within the limited amount of memory available.

The usual way of dealing with programs that exceed resource limits is to monitor their usage and abort execution when limits are exceeded. Apart from the waste that this entails – including the resources consumed by the monitoring itself – it necessitates programming recovery action in the case of failure.

The Mobile Resource Guarantees (MRG) project is applying ideas from proof-carrying code to the problem of resource certification for mobile code. As with other work on proof-carrying code for safety properties, certificates contain

formal proofs, but in our case, they claim a resource usage property. Work in MRG has so far concentrated mainly on bounds on heap space usage, but most of the infrastructure that has been built is reusable for bounds on other kinds of resources. One difference between MRG and other work on proof-carrying code is that proof certificates in MRG refer to bytecode programs rather than native code. One bytecode language of particular interest is JVML [22] but there are others, including the CIL bytecode of the Microsoft .NET framework [24], JavaCard [33], and the restricted version of JVML described in [32]. An elegant solution to the tension between the engineering requirement to make theorem proving and proof checking tractable, while at the same time remaining faithful to the imperative semantics of these underlying bytecode languages, is the Grail intermediate language (see Sect. 5) which also targets multiple bytecode languages.

One of the central issues in work on proof-carrying code is how proofs of properties of code are produced. One traditional approach is for object code and proofs to be generated from source code in a high-level language by a *certifying compiler* like Touchstone [10], using types and other high-level source information[1]. The MRG project follows this approach, building on innovative work on linear resource-aware type systems [14, 15], whereby programs are certified by virtue of their typing as satisfying certain resource bounds. For instance, in a space-aware type system, the type of an in-place sorting function would be different from the type of a sorting function, like merge sort, that requires extra working space to hold a copy of its input; still different would be the type of a sorting function that requires a specific number of extra cells to do its work, independent of the size of its input. A corresponding proof of this behaviour at the bytecode level can be generated automatically from a typing derivation in such a system in the course of compiling the program to bytecode. It even turns out to be possible to *infer* heap space requirements in many situations [16]. This work has been carried out in a first-order ML-like functional language, Camelot (described in Sect. 3), that has been developed as a testbed by the MRG project. The underlying proof-carrying code infrastructure operates at the bytecode (Grail) level; Camelot is just an example of a language that a code producer might use to produce bytecode together with a proof that it satisfies some desired resource bound.

This paper is an overview of the achievements of the MRG project as of the summer of 2004. It is self-contained, but due to space limitations many points are sketched or glossed over; full technical details can be found in the papers that are cited below. The main contribution of the paper is a presentation of the overall picture into which these technical contributions are meant to fit.

In the next section, we describe the overall architecture of the MRG framework, including the rôle of the two language levels (Grail and Camelot), and how MRG-style proof-carrying code fits with standard Java security. Sections 3 and 4

[1] A slightly different approach was taken by the work on Typed Assembly Language ([25] and later), where a fixed type system is provided for the low-level language, and certification amounts to providing a typing in this low-level type system.

focus on the "upper" language level, introducing Camelot and space-aware type systems. Section 5 focuses on the "lower" language level, describing the Grail intermediate language and the way that it provides both a tractable basis for proof and relates to (multiple) imperative bytecode languages. Section 6 ties the two language levels together by explaining the logic for expressing proofs of resource properties of bytecode programs and the generation of proofs from resource typings. A conclusion outlines the current status of the MRG project and summarizes its contributions.

2 Architecture and Deployment

In this section we discuss the architecture of a smart device-based system which deploys the technology of the MRG project in a novel protocol for certifying resource bounds on downloaded code from an untrusted source. Our protocol is designed so that it can be integrated with the built-in mechanism for Java bytecode checking, via the *Security Manager*. In the JVM, the Security Manager is entrusted with enforcing the security policy designated by the user, and ensuring that no violations of the security policy occur while the code runs.

In our protocol, a *Resource Manager* is responsible for verifying that the certificate supplied with a piece of code ensures that it will execute within the advertised resource constraints. A *Proof Checker* is invoked to do this. If the check succeeds, we have an absolute guarantee that the resource bounds are met, so it is not necessary to check for resource violations as the code runs. Our Resource Manager is not a replacement for the standard Java Security Manager but instead forms a *perimeter defense* which prevents certain non-well-behaved programs from being executed at all.

The Mobile Resource Guarantees framework provides a high-level language, Camelot, and a low-level language, Grail, into which this is compiled. (Camelot is presented in more detail in Sect. 3 and Grail is discussed in Sect. 5.) Application developers work in the high-level language and interact with resource typing judgements at the appropriate level of abstraction for their realm of expertise. For this approach to be successful it is necessary for the compilation process to be *transparent* [23] in that the resource predictions made at the high-level language level must survive the compilation process so that they remain true at the low level. This places constraints on the expressive power of the high-level language, prohibiting the inclusion of some more complex language features. It also places constraints on the nature of the compilation process itself, requiring the compiler to sometimes sacrifice peak efficiency for predictability, which is the familiar trade-off from development of real-time software.

A consumer of proof-carrying code (such as Grail class files with attached proofs of resource consumption) requires an implementation technology which enforces the security policy that they specify. The *Java agents* introduced in the J2SDK version 1.5.0 provide the most direct way to implement these policies. An agent is a "hook" in the JVM allowing the PCC consumer to attach their own implementation of their security policy as an instance of a general-purpose PCC Security Manager.

Java agents can be used for several resource-bound-specific purposes:

1. to query the attached proof and decide to refuse to load, build and execute the class if necessary;
2. to apply *per-class* or *per-package* use restrictions by modifying each method in the class with entry and exit assertions that inspect resource consumption measures; and
3. to apply *per-method* constraints on heap-allocation and run-time by instrumenting method bodies.

Each of these checks can be unloaded at JVM instantiation time to allow a mobile-code consumer to vary their security policy between its tightest and laxest extrema.

3 Space Types and Camelot

This section describes the high-level language Camelot and the space type system which together allow us to produce JVM bytecode endowed with guaranteed and certified bounds on heap space consumption.

Syntactically, and as far as its functional semantics is concerned, Camelot is essentially a fragment of the ML dialect O'Caml [29]. In particular, it provides the usual recursive datatypes and recursive (not necessarily primitive recursive) definition of functions using pattern matching, albeit restricted to flat patterns.

One difference to O'Caml is that Camelot compiles to JVM bytecode and provides (via the O'Camelot extension [36]) a smooth integration of genuine Java methods and objects.

The most important difference, however, lies in Camelot's memory model. This uses a freelist, managed directly by the compiled code, rather than relying exclusively on garbage collection. All non-primitive types in a Camelot program are compiled to JVM objects of a single class Diamond, which contains appropriate fields to hold data for a single node of any datatype. Unused objects are released to the freelist so that their space can be immediately reused. The compiler generates the necessary code to manage the freelist, based on some language annotations described below.

This conflation of types into a single allocation unit is standard for memory recycling in constrained environments; there is some loss of space around the edges, but management is simple and in our case formally guaranteed to succeed. If required, we could duplicate our analysis to manage a range of cell sizes in parallel, but we have not yet seen compelling examples for this.

3.1 The Diamond Type

Following [14], Camelot has an abstract type denoted <> whose members are heap addresses of Diamond-objects. The only way to access this type is via datatype constructors. Suppose for example that we have defined a type of integer lists as follows[2]

[2] The annotation ! ensures that the constructor Nil is represented by a null pointer rather than a proper object.

```
type iList = !Nil | Cons of int * iList
```

If this is the only type occurring in a program then the Diamond class will look as follows (in simplified form and Java notation):

```
public class Diamond extends java.lang.Object {
    public Diamond R0;
    public int V1;
}
```

If, say, x1 is an element of type iList, hence compiled to an object reference of type Diamond, we can form a new list x2 by

```
let x2 = Cons(9,x1) in ...
```

The required object reference will be taken from the aforementioned freelist providing it is non-empty. Otherwise, the JVM new instruction will be executed to allocate a new object of type Diamond.

If, however, we have in our local context an element d of type <> then we can alternatively form x2 by

```
let x2 = Cons(9,x1)@d in ...
```

thus instructing the compiler to put the new Cons cell into the Diamond object referenced by d, whose contents will be overwritten.

Using these phrases in the context of pattern matching provides us with elements of type <> and also refills the freelist. A pattern match like

```
match x with
  Cons(h,t)@d -> ...
```

is evaluated by binding h, t and d to the contents of the "head" (h) and "tail" (t) fields and the reference to x itself (d). Thus, in the body of the pattern match d is an element of type <> available for constructing new Cons cells.

Alternatively, the syntax

```
match x with
  Cons(h,t)@_ -> ...
```

returns the cell occupied by x to the freelist for later use.

Finally, an unannotated pattern match such as

```
match x with
  Cons(h,t) -> ...
```

performs ordinary non-destructive matching.

3.2 Linear Typing

When a list x is matched against a pattern of the form Cons(h,t)@d or Cons(h,t)@_ it is the responsibility of the programmer to ensure that the list x itself is not

used anymore because its contents will be overwritten subsequently. For this purpose, the Camelot compiler has an option that enforces (affine) linear use of all variables. If all variables are used at most once in their scope then there can in particular be no reference to x in the body of the pattern match above. In [14] a formal proof is given that such a program behaves purely functionally, i.e., as if the type <> was replaced by the unit type. Linear typing is, however, a fairly crude discipline and rules out many sound programs. In [6] we present an improved type system that distinguishes between modifying and read-only access to a data structure and in particular allows multiple read-only accesses, which would be ruled out by the linear discipline. This is not yet implemented in Camelot. Alternatively, the programmer can turn off the linear typing option and rely on his or her own judgement, or use some other scheme.

3.3 Extended Example

The code in Figure 1 shows a standalone Camelot application containing a function start : string list −> unit which serves as an entry point. It is assumed that the program is executed by applying start to an (ordinary) list of strings obtained, e.g., from the standard input.

We see that the function ins destroys its argument, whereas the sorting function sort : ilist −> ilist, as well as the display function show_list : ilist −> unit, each leave their argument intact.

3.4 Certification of Memory Usage

The idea behind certification of heap-space usage in MRG is as follows: given a Camelot program containing a function start : string list −> unit, find a linear function $s(x) = ax + b$ with the property that evaluating (the compiled version of) start on an input list of length n will not invoke the new instruction *provided* that the freelist contains initially no less than $s(n)$ cells.

Once we have such a linear function s we can then package our compiled bytecode together with a wrapper that takes input from stdin or a file, initialises (using new) the freelist to hold $s(n)$ cells where n is the size of the input, and then evaluates start.

3.5 Inference of Space Bounds

Such linear space bounds can efficiently be obtained using the type-based analysis described in [16] which has subsequently been implemented and tuned to Camelot in [17]. In summary, this analysis infers for each function contained in the program a numerically annotated type describing its space usage. The desired bounding function can then be directly read off from the type of start.

The result of running the analysis on our example program is given in Figure 2. The entry

```
ins : 1, int −> iList[0|int,#,0] −> iList[0|int,#,0], 0;
      ↑
```