Roberto Gorrieri
Heike Wehrheim (Eds.)

LNCS 4037

Formal Methods for Open Object-Based Distributed Systems

8th IFIP WG 6.1 International Conference, FMOODS 2006
Bologna, Italy, June 2006
Proceedings

ifip

Springer

Roberto Gorrieri Heike Wehrheim (Eds.)

Formal Methods for Open Object-Based Distributed Systems

8th IFIP WG 6.1 International Conference, FMOODS 2006
Bologna, Italy, June 14-16, 2006
Proceedings

Springer

Volume Editors

Roberto Gorrieri
Università di Bologna
Dipartimento di Scienze dell'Informazione
Mura A. Zamboni, 7, 40127 Bologna, Italy
E-mail: gorrieri@cs.unibo.it

Heike Wehrheim
Universität Paderborn
Institut für Informatik
Warburger Str. 100, 33098 Paderborn, Germany
E-mail: wehrheim@uni-paderborn.de

Library of Congress Control Number: 2006926884

CR Subject Classification (1998): C.2.4, D.1.3, D.2, D.3, F.3, D.4

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-34893-X Springer Berlin Heidelberg New York
ISBN-13 978-3-540-34893-1 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11768869 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 4037

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

Lecture Notes in Computer Science

For information about Vols. 1–3921

please contact your bookseller or Springer

Vol. 4039: M. Morisio (Ed.), Reuse of Off-the-Shelf Components. XIII, 444 pages. 2006.

Vol. 4038: P. Ciancarini, H. Wiklicky (Eds.), Coordination Models and Languages. VIII, 299 pages. 2006.

Vol. 4037: R. Gorrieri, H. Wehrheim (Eds.), Formal Methods for Open Object-Based Distributed Systems. X, 267 pages. 2006.

Vol. 4034: J. Münch, M. Vierimaa (Eds.), Product-Focused Software Process Improvement. XVII, 474 pages. 2006.

Vol. 4027: H.L. Larsen, G. Pasi, D. Ortiz-Arroyo, T. Andreasen, H. Christiansen (Eds.), Flexible Query Answering Systems. XVIII, 714 pages. 2006. (Sublibrary LNAI).

Vol. 4024: S. Donatelli, P. S. Thiagarajan (Eds.), Petri Nets and Other Models of Concurrency - ICATPN 2006. XI, 441 pages. 2006.

Vol. 4021: E. André, L. Dybkjær, W. Minker, H. Neumann, M. Weber (Eds.), Perception and Interactive Technologies. XI, 217 pages. 2006. (Sublibrary LNAI).

Vol. 4011: Y. Sure, J. Domingue (Eds.), The Semantic Web: Research and Applications. XIX, 726 pages. 2006.

Vol. 4010: S. Dunne, B. Stoddart (Eds.), Unifying Theories of Programming. VIII, 257 pages. 2006.

Vol. 4007: C. Àlvarez, M. Serna (Eds.), Experimental Algorithms. XI, 329 pages. 2006.

Vol. 4006: L.M. Pinho, M. González Harbour (Eds.), Reliable Software Technologies – Ada-Europe 2006. XII, 241 pages. 2006.

Vol. 4004: S. Vaudenay (Ed.), Advances in Cryptology - EUROCRYPT 2006. XIV, 613 pages. 2006.

Vol. 4003: Y. Koucheryavy, J. Harju, V.B. Iversen (Eds.), Next Generation Teletraffic and Wired/Wireless Advanced Networking. XVI, 582 pages. 2006.

Vol. 4001: E. Dubois, K. Pohl (Eds.), Advanced Information Systems Engineering. XVI, 560 pages. 2006.

Vol. 3999: C. Kop, G. Fliedl, H.C. Mayr, E. Métais (Eds.), Natural Language Processing and Information Systems. XIII, 227 pages. 2006.

Vol. 3998: T. Calamoneri, I. Finocchi, G.F. Italiano (Eds.), Algorithms and Complexity. XII, 394 pages. 2006.

Vol. 3997: W. Grieskamp, C. Weise (Eds.), Formal Approaches to Software Testing. XII, 219 pages. 2006.

Vol. 3996: A. Keller, J.-P. Martin-Flatin (Eds.), Self-Managed Networks, Systems, and Services. X, 185 pages. 2006.

Vol. 3995: G. Müller (Ed.), Emerging Trends in Information and Communication Security. XX, 524 pages. 2006.

Vol. 3994: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2006, Part IV. XXXV, 1096 pages. 2006.

Vol. 3993: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2006, Part III. XXXVI, 1136 pages. 2006.

Vol. 3992: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2006, Part II. XXXV, 1122 pages. 2006.

Vol. 3991: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science – ICCS 2006, Part I. LXXXI, 1096 pages. 2006.

Vol. 3990: J. C. Beck, B.M. Smith (Eds.), Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. X, 301 pages. 2006.

Vol. 3989: J. Zhou, M. Yung, F. Bao, Applied Cryptography and Network Security. XIV, 488 pages. 2006.

Vol. 3987: M. Hazas, J. Krumm, T. Strang (Eds.), Location- and Context-Awareness. X, 289 pages. 2006.

Vol. 3986: K. Stølen, W.H. Winsborough, F. Martinelli, F. Massacci (Eds.), Trust Management. XIV, 474 pages. 2006.

Vol. 3984: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part V. XXV, 1045 pages. 2006.

Vol. 3983: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part IV. XXVI, 1191 pages. 2006.

Vol. 3982: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part III. XXV, 1243 pages. 2006.

Vol. 3981: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part II. XXVI, 1255 pages. 2006.

Vol. 3980: M. Gavrilova, O. Gervasi, V. Kumar, C.J. K. Tan, D. Taniar, A. Laganà, Y. Mun, H. Choo (Eds.), Computational Science and Its Applications - ICCSA 2006, Part I. LXXV, 1199 pages. 2006.

Vol. 3979: T.S. Huang, N. Sebe, M.S. Lew, V. Pavlović, M. Kölsch, A. Galata, B. Kisačanin (Eds.), Computer Vision in Human-Computer Interaction. XII, 121 pages. 2006.

Vol. 3978: B. Hnich, M. Carlsson, F. Fages, F. Rossi (Eds.), Recent Advances in Constraints. VIII, 179 pages. 2006. (Sublibrary LNAI).

Vol. 3976: F. Boavida, T. Plagemann, B. Stiller, C. Westphal, E. Monteiro (Eds.), Networking 2006. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems. XXVI, 1276 pages. 2006.

Vol. 3975: S. Mehrotra, D.D. Zeng, H. Chen, B. Thuraisingham, F.-Y. Wang (Eds.), Intelligence and Security Informatics. XXII, 772 pages. 2006.

Vol. 3973: J. Wang, Z. Yi, J.M. Zurada, B.-L. Lu, H. Yin (Eds.), Advances in Neural Networks - ISNN 2006, Part III. XXIX, 1402 pages. 2006.

Vol. 3972: J. Wang, Z. Yi, J.M. Zurada, B.-L. Lu, H. Yin (Eds.), Advances in Neural Networks - ISNN 2006, Part II. XXVII, 1444 pages. 2006.

Vol. 3971: J. Wang, Z. Yi, J.M. Zurada, B.-L. Lu, H. Yin (Eds.), Advances in Neural Networks - ISNN 2006, Part I. LXVII, 1442 pages. 2006.

Vol. 3970: T. Braun, G. Carle, S. Fahmy, Y. Koucheryavy (Eds.), Wired/Wireless Internet Communications. XIV, 350 pages. 2006.

Vol. 3968: K.P. Fishkin, B. Schiele, P. Nixon, A. Quigley (Eds.), Pervasive Computing. XV, 402 pages. 2006.

Vol. 3967: D. Grigoriev, J. Harrison, E.A. Hirsch (Eds.), Computer Science – Theory and Applications. XVI, 684 pages. 2006.

Vol. 3966: Q. Wang, D. Pfahl, D.M. Raffo, P. Wernick (Eds.), Software Process Change. XIV, 356 pages. 2006.

Vol. 3965: M. Bernardo, A. Cimatti (Eds.), Formal Methods for Hardware Verification. VII, 243 pages. 2006.

Vol. 3964: M. Ü. Uyar, A.Y. Duale, M.A. Fecko (Eds.), Testing of Communicating Systems. XI, 373 pages. 2006.

Vol. 3963: O. Dikenelli, M.-P. Gleizes, A. Ricci (Eds.), Engineering Societies in the Agents World VI. X, 303 pages. 2006. (Sublibrary LNAI).

Vol. 3962: W. IJsselsteijn, Y. de Kort, C. Midden, B. Eggen, E. van den Hoven (Eds.), Persuasive Technology. XII, 216 pages. 2006.

Vol. 3960: R. Vieira, P. Quaresma, M.d.G.V. Nunes, N.J. Mamede, C. Oliveira, M.C. Dias (Eds.), Computational Processing of the Portuguese Language. XII, 274 pages. 2006. (Sublibrary LNAI).

Vol. 3959: J.-Y. Cai, S. B. Cooper, A. Li (Eds.), Theory and Applications of Models of Computation. XV, 794 pages. 2006.

Vol. 3958: M. Yung, Y. Dodis, A. Kiayias, T. Malkin (Eds.), Public Key Cryptography - PKC 2006. XIV, 543 pages. 2006.

Vol. 3956: G. Barthe, B. Grégoire, M. Huisman, J.-L. Lanet (Eds.), Construction and Analysis of Safe, Secure, and Interoperable Smart Devices. IX, 175 pages. 2006.

Vol. 3955: G. Antoniou, G. Potamias, C. Spyropoulos, D. Plexousakis (Eds.), Advances in Artificial Intelligence. XVII, 611 pages. 2006. (Sublibrary LNAI).

Vol. 3954: A. Leonardis, H. Bischof, A. Pinz (Eds.), Computer Vision – ECCV 2006, Part IV. XVII, 613 pages. 2006.

Vol. 3953: A. Leonardis, H. Bischof, A. Pinz (Eds.), Computer Vision – ECCV 2006, Part III. XVII, 649 pages. 2006.

Vol. 3952: A. Leonardis, H. Bischof, A. Pinz (Eds.), Computer Vision – ECCV 2006, Part II. XVII, 661 pages. 2006.

Vol. 3951: A. Leonardis, H. Bischof, A. Pinz (Eds.), Computer Vision – ECCV 2006, Part I. XXXV, 639 pages. 2006.

Vol. 3950: J.P. Müller, F. Zambonelli (Eds.), Agent-Oriented Software Engineering VI. XVI, 249 pages. 2006.

Vol. 3947: Y.-C. Chung, J.E. Moreira (Eds.), Advances in Grid and Pervasive Computing. XXI, 667 pages. 2006.

Vol. 3946: T.R. Roth-Berghofer, S. Schulz, D.B. Leake (Eds.), Modeling and Retrieval of Context. XI, 149 pages. 2006. (Sublibrary LNAI).

Vol. 3945: M. Hagiya, P. Wadler (Eds.), Functional and Logic Programming. X, 295 pages. 2006.

Vol. 3944: J. Quiñonero-Candela, I. Dagan, B. Magnini, F. d'Alché-Buc (Eds.), Machine Learning Challenges. XIII, 462 pages. 2006. (Sublibrary LNAI).

Vol. 3943: N. Guelfi, A. Savidis (Eds.), Rapid Integration of Software Engineering Techniques. X, 289 pages. 2006.

Vol. 3942: Z. Pan, R. Aylett, H. Diener, X. Jin, S. Göbel, L. Li (Eds.), Technologies for E-Learning and Digital Entertainment. XXV, 1396 pages. 2006.

Vol. 3941: S.W. Gilroy, M.D. Harrison (Eds.), Interactive Systems. XI, 267 pages. 2006.

Vol. 3940: C. Saunders, M. Grobelnik, S. Gunn, J. Shawe-Taylor (Eds.), Subspace, Latent Structure and Feature Selection. X, 209 pages. 2006.

Vol. 3939: C. Priami, L. Cardelli, S. Emmott (Eds.), Transactions on Computational Systems Biology IV. VII, 141 pages. 2006. (Sublibrary LNBI).

Vol. 3936: M. Lalmas, A. MacFarlane, S. Rüger, A. Tombros, T. Tsikrika, A. Yavlinsky (Eds.), Advances in Information Retrieval. XIX, 584 pages. 2006.

Vol. 3935: D. Won, S. Kim (Eds.), Information Security and Cryptology - ICISC 2005. XIV, 458 pages. 2006.

Vol. 3934: J.A. Clark, R.F. Paige, F.A. C. Polack, P.J. Brooke (Eds.), Security in Pervasive Computing. X, 243 pages. 2006.

Vol. 3933: F. Bonchi, J.-F. Boulicaut (Eds.), Knowledge Discovery in Inductive Databases. VIII, 251 pages. 2006.

Vol. 3931: B. Apolloni, M. Marinaro, G. Nicosia, R. Tagliaferri (Eds.), Neural Nets. XIII, 370 pages. 2006.

Vol. 3930: D.S. Yeung, Z.-Q. Liu, X.-Z. Wang, H. Yan (Eds.), Advances in Machine Learning and Cybernetics. XXI, 1110 pages. 2006. (Sublibrary LNAI).

Vol. 3929: W. MacCaull, M. Winter, I. Düntsch (Eds.), Relational Methods in Computer Science. VIII, 263 pages. 2006.

Vol. 3928: J. Domingo-Ferrer, J. Posegga, D. Schreckling (Eds.), Smart Card Research and Advanced Applications. XI, 359 pages. 2006.

Vol. 3927: J. Hespanha, A. Tiwari (Eds.), Hybrid Systems: Computation and Control. XII, 584 pages. 2006.

Vol. 3925: A. Valmari (Ed.), Model Checking Software. X, 307 pages. 2006.

Vol. 3924: P. Sestoft (Ed.), Programming Languages and Systems. XII, 343 pages. 2006.

Vol. 3923: A. Mycroft, A. Zeller (Eds.), Compiler Construction. XIII, 277 pages. 2006.

Vol. 3922: L. Baresi, R. Heckel (Eds.), Fundamental Approaches to Software Engineering. XIII, 427 pages. 2006.

Preface

This volume contains the proceedings of the 8th IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS 2006). The conference was held in Bologna, Italy, 14-16 June 2006, as part of the federated multiconference DisCoTec (Distributed Computing Techniques), together with the 8th International Conference on Coordination Models and Languages (COORDINATION) and the 6th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS). DisCoTec was organized by the Department of Computer Science of the University of Bologna.

Established in 1996, the FMOODS series of conferences aims to provide an integrated forum for research on formal aspects of open object-based distributed systems. The FMOODS 2006 especially attracted novel contributions reflecting recent developments in the area, such as component- and model-based design, service-oriented computing, and software quality. Some more specific topics of interest were: semantics and implementation of object-oriented programming and (visual) modelling languages; formal techniques for specification, design, analysis, verification, validation and testing; formal methods for service-oriented computing; and integration of quality of service requirements into formal models.

These proceedings contain a selection of 16 research contributions, out of 51 submissions, which went through a rigorous review process by international reviewers. We therefore owe special thanks to all members of the Program Committee, and their sub-referees, for the excellent work they have done in the short time they had.

Additionally, these proceedings include three invited papers by Pierpaolo Degano (University of Pisa), José Luiz Fiadeiro (University of Leicester) and Davide Sangiorgi (University of Bologna).

Finally, our thanks go to the Organizing Committee of the DisCoTec federated conference, chaired by Gianluigi Zavattaro, for the excellent work done and for the support they gave in managing the submission system by Philippe Rigaux. We also gratefully acknowledge the financial support of the Department of Computer Science of the University of Bologna and from the EU-project SENSORIA.

June 2006

Roberto Gorrieri
Heike Wehrheim

Organization

General Chair	Gianluigi Zavattaro (University of Bologna, Italy)
Program Chairs	Roberto Gorrieri (University of Bologna, Italy)
	Heike Wehrheim (University of Paderborn, Germany)
Publicity Chair	Martin Steffen (University of Kiel, Germany)

Steering Committee

John Derrick (University of Sheffield, UK)
Roberto Gorrieri (University of Bologna, Italy)
Elie Najm (ENST, Paris, France)

Program Committee

Lynne Blair (U. of Lancaster, UK)
Eerke Boiten (U. of Kent, UK)
Nadia Busi (U. of Bologna, Italy)
John Derrick (U. of Sheffield, UK)
Alessandro Fantechi (U. of Florence, Italy)
Colin Fidge (U. of Queensland, Australia)
Robert France (Colorado State U., USA)
Roberto Gorrieri (U. of Bologna, Italy)
Reiko Heckel (U. of Leicester, UK)
Einar Broch Johnsen (U. of Oslo, Norway)
Doug Lea (State U. of New York, USA)
Elie Najm (ENST Paris, France)
Uwe Nestmann (TU Berlin, Germany)
Erik Poll (U. of Nijmegen, Netherlands)
Arend Rensink (U. Twente, Netherlands)
Ralf Reussner (U. of Karlsruhe, Germany)
Bernhard Rumpe (TU Braunschweig, Germany)
Martin Steffen (U. of Kiel, Germany)
Carolyn Talcott (SRI International, USA)
Andrzej Tarlecki (Warsaw University, Poland)
Vasco Vasconcelos (U. of Lisbon, Portugal)
Heike Wehrheim (U. of Paderborn, Germany)
Elena Zucca (U. of Genova, Italy)

Organizing Committee

Claudio Guidi
Ivan Lanese
Roberto Lucchi
Luca Padovani
Elisa Turrini
Stefano Zacchiroli

Referees

Davide Ancona
Michele Banci
Laura Bocchi
Edoardo Bonta
Mario Bravetti
Manuel Breschi
Barbara Catania
Walter Cazzola
Antonio Cerone
Giorgio Delzanno
Piergiorgio Di Giacomo
Luca Durante
Karsten Ehrig
Harald Fecher
Maurizio Gabbrielli
Geri Georg
Hans Grönniger
Andreas Gruener
Christian Haack
Jan Hendrik Hausmann
Marcel Kyas
Giovanni Lagorio
Grzegorz Marczyński
Francisco Martins
Viviana Mascardi
Peter lveczky
Wiesław Pawłowski
Holger Rasch
Antonio Ravara
Dirk Reiß
Paolo Rosso
Murat Sahingöz
Luigi Sassoli
Martin Schindler
Gerardo Schneider
Aleksy Schubert
Graeme Smith
Mark Stein
Gabriele Taentzer
Simone Tini
Hugo Vieira
Steven Voelkel
Gianluigi Zavattaro
Artur Zawłocki

Table of Contents

I Invited Speakers

Security Issues in Service Composition
Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari 1

Separating Distribution from Coordination and Computation as Architectural Dimensions
José Luiz Fiadeiro 17

The Bisimulation Proof Method: Enhancements and Open Problems
Davide Sangiorgi 18

II Regular Papers

An Approach to Quality Achievement at the Architectural Level: AQUA
Heeseok Choi, Keunhyuk Yeom, Youhee Choi, Mikyeong Moon 20

Bounded Analysis and Decomposition for Behavioural Descriptions of Components
Pascal Poizat, Jean-Claude Royer, Gwen Salaün 33

Modeling and Validation of a Software Architecture for the Ariane-5 Launcher
Iulian Ober, Susanne Graf, David Lesens 48

Synchronizing Behavioural Mismatch in Software Composition
Carlos Canal, Pascal Poizat, Gwen Salaün 63

Static Safety for an Actor Dedicated Process Calculus by Abstract Interpretation
Pierre-Loïc Garoche, Marc Pantel, Xavier Thirioux 78

Temporal Superimposition of Aspects for Dynamic Software Architecture
Carlos E. Cuesta, María del Pilar Romay, Pablo de la Fuente, Manuel Barrio-Solórzano 93

Modeling Long–Running Transactions with Communicating Hierarchical Timed Automata
Ruggero Lanotte, Andrea Maggiolo-Schettini, Paolo Milazzo, Angelo Troina 108

Transformation Laws for UML-RT
Rodrigo Ramos, Augusto Sampaio, Alexandre Mota 123

Underspecification, Inherent Nondeterminism and Probability in Sequence Diagrams
Atle Refsdal, Ragnhild Kobro Runde, Ketil Stølen 138

Generating Instance Models from Meta Models
Karsten Ehrig, Jochen M. Küster, Gabriele Taentzer, Jessica Winkelmann 156

KM3: A DSL for Metamodel Specification
Frédéric Jouault, Jean Bézivin 171

Defining Object-Oriented Execution Semantics Using Graph Transformations
Harmen Kastenberg, Anneke Kleppe, Arend Rensink 186

Type-Safe Runtime Class Upgrades in Creol
Ingrid Chieh Yu, Einar Broch Johnsen, Olaf Owe 202

Abstract Interface Behavior of Object-Oriented Languages with Monitors
Erika Ábrahám, Andreas Grüner, Martin Steffen 218

Mobility Mechanisms in Service Oriented Computing
Claudio Guidi, Roberto Lucchi 233

Theoretical Foundations of Scope-Based Compensable Flow Language for Web Service
Geguang Pu, Huibiao Zhu, Zongyan Qiu, Shuling Wang, Xiangpeng Zhao, Jifeng He 251

Author Index 267

Security Issues in Service Composition

Massimo Bartoletti, Pierpaolo Degano, and Gian Luigi Ferrari

Dipartimento di Informatica, Università di Pisa, Italy
{bartolet, degano, giangi}@di.unipi.it

Abstract. We use a distributed, enriched λ-calculus for describing networks of services. Both services and their clients can protect themselves, by imposing security constraints on each other's behaviour. Then, service interaction results in a call-by-property mechanism, that matches the client requests with service's. A static approach is also described, that determines how to compose services while guaranteeing that their execution is always secure, without resorting to any dynamic check.

1 Introduction

Service-oriented computing (SOC) is an emerging paradigm to design distributed applications [31, 30, 19]. In this paradigm, applications are built by assembling together independent computational units, called *services*. A service is a standalone component distributed over a network, and made available through standard interaction mechanisms. An important aspect is that services are *open*, in that they are built with little or no knowledge about their operating environment, their clients, and further services therein invoked. Composition of services may require peculiar mechanisms to handle complex interaction patterns (e.g. to implement transactions), while enforcing non-functional requirements on the system behaviour (e.g. security and service level agreement). Web Services [3, 34, 38] built upon XML technologies are possibly the most illustrative and well developed example of the SOC paradigm. Indeed, a variety of XML-based technologies already exists for describing, discovering and invoking web services [18, 14, 5, 39]. There are also several standards for defining and enforcing non-functional requirements of services, e.g. WS-Security [6], WS-Trust [4] and WS-Policy [15] among the others.

1.1 Security and Service Composition

The *orchestration* of services consists of their composition and coordination. Languages for that have been recently proposed, e.g. BPEL4WS [5, 25]. Service composition heavily depends on which information about a service is made public, on how to choose those services that match the user's requirements, and on their actual run-time behaviour. Security makes service composition even harder. Services may be offered by different providers, which only partially trust each other. On the one hand, providers have to guarantee the delivered service to respect a given security policy, in any interaction with the operational

R. Gorrieri and H. Wehrheim (Eds.): FMOODS 2006, LNCS 4037, pp. 1–16, 2006.

environment, and regardless of who actually called the service. On the other hand, clients may want to protect their sensible data from the services invoked.

A typical approach consists in endowing the network infrastructure with authentication mechanisms, so to certify the identity of services. However, security may be breached even by trusted services, either because of unintentional behaviour (e.g. bugs), or because the composition of the client and the services exhibits some behaviour unwanted by the client (e.g. leakage of information).

We have addressed the problem of security in a linguistic framework. In our approach, clients may protect from their callers by wrapping security-critical portions of their own code into *safety framings*. These framings enforce the given security policy on the execution of the wrapped piece of code, aborting it whenever about to violate the policy, thus offering additional flexibility with respect to monolithic *global* policies, and relieving the programmer of guarding each use of security-critical resources.

On their side, callers may constrain the behaviour of the called services, by supplying a security policy at the moment of invocation. We push further this invocation mechanism, by allowing callers to request services that not only do obey the imposed security constraints, but that also respect a given contract on their functional behaviour. The implementation of this so-called *call-by-property* invocation mechanism requires that services are published together with a *certified* abstraction of their behaviour.

1.2 The Planning Problem

Call-by-property invocation and safety framings make service composition secure. A *plan* orchestrates the execution of a service-based application, by associating the sequence of run-time service requests with a corresponding sequence of selected services. A major problem is still left open: how to construct a plan that guarantees no executions will abort because of some action attempting to violate security.

Determining such a viable plan amounts to selecting from the network those services that accomplish the requested task, while respecting the security constraints on demand. Those services that locally obey the property imposed by a request are not always good candidates, because their behaviour may affect security of the whole composition. For example, consider a device with a limited computational power. Suppose it downloads an applet from the network, and then delegates a remote service to run it. Although the contract between the device and the code provider is fulfilled, the applet may violate a security policy imposed by the executer. To determine the viable plans, one has to check the effects of all the available applets against the security policies of all the remote executers.

As a matter of fact, there might be several different kinds of plans, each with a different expressive power. Among them, one may consider plans that attach a selection of services to each program point representing a service request. The expressive power varies according to the nature of the information associated with each request. *Simple plans* associate a single service with each request,

multi-choice plans map requests into sets of services, and *dependent plans* also convey the dependence of a service selection with the choices made in the past (a sort of continuation-passing plan). These kinds of plans have been studied in [9]. *Dependent multi-choice* plans are a mix of the last two kinds. Further expressive power is gained when relaxing the assumption of associating service selections to the program points where requests are made. *Regular plans* drive the execution of a program, by providing it with the possible patterns of service selections, in the form of a regular expression. *Dynamic plans* can be updated at run-time, according to the evaluation of some conditions on the program execution (e.g. boolean guards in conditionals, number of iterations in a loop, etc.).

1.3 A Static Approach to Secure Service Composition

We have proposed a solution to the planning problem, within a distributed framework [10]. Services are functional units in an enriched λ-calculus, they are explicitly located at network sites, and they have a published public interface. Unlike standard syntactic signatures, this interface includes an abstraction of the service behaviour, in the form of annotated types. To obtain a service with a specific behaviour, a client queries the network for a published interface matching the requirements. Security is implemented by wrapping the critical blocks of code inside safety framings (with local scopes, possibly nested), that enforce the relevant policies during the execution of the block. In the spirit of history-based security [1], a security policy can inspect the whole execution history at a given site. Since our framework is fully distributed, our policies cannot span over multiple sites.

We have introduced a type and effect system for our calculus [21, 28, 35]. The type of a service describes its functional behaviour, while the effect is a *history expression*, representing those histories of events relevant to security. History expressions extend regular expressions with information about the selection of services, coupled with their corresponding effect.

We have then devised a way of extracting from a history expression all the *viable* plans, i.e. those that successfully drive secure executions. This is a two-stage construction. A first transformation of history expressions makes them model-checkable for validity [7]. Valid history expressions guarantee that the services they come from never go wrong at run-time. From valid histories it is then immediate to obtain the viable plans, that make any execution monitor unneeded.

1.4 Trusted Orchestration

Our planning technique acts as a *trusted orchestrator* of services. It provides a client with the plans guaranteeing that the invoked services always respect the required properties. Thus, in our framework the only trusted entity is the orchestrator, and neither clients nor services need to be such. In particular, the orchestrator infers functional and behavioural types of each service. Also, it is responsible for certifying the service code, for publishing its interface, and for

guaranteeing that services will not arbitrarily change their code on the fly: when this happens, services need to be certified again.

When an application is injected in the network, the orchestrator provides it with a viable plan (if any), constructed by composing and analysing the certified interfaces of the available services. The trustworthiness of the orchestrator relies upon formal grounds. We proved the soundness of our type and effect system, and the correctness of the static analysis and model-checking technique that infers viable plans.

The orchestrator constructs the plans for a client, by considering the view of the network at the moment the application is injected. To be more dynamic, one would like to manage the discovering of new services, as well as the case when existing ones are no longer available.

Both these problems require a special treatment. Multi-choice plans are a first solution to deal with disappearing services, because they offer many choices for the same request. Publication of new services poses instead a major problem. To cope with that, one has to reconfigure plans at run-time, by exploiting the new interfaces. However, incrementally checking viability of plans is an open problem. A possible solution is to enrich history expressions with *hooks* where new services can be attached. The orchestrator then needs to check the validity of the newly discovered plans, hopefully in an incremental manner.

1.5 Related Work

The secure composition of components underlies the design of Sewell and Vitek's box-π [33], an extension of the π-calculus that allows for expressing safety policies in the form of *security wrappers*. These are programs that encapsulate a component to control the interactions with other (possibly untrusted) components. A type system that statically captures the allowed causal information flows between components. Our safety framings are closely related to wrappers.

Gorla, Hennessy and Sassone [23] consider a calculus for agents which may migrate between sites in a controlled manner. Each site has a *membrane*, representing both a security policy and a classification of the levels of trust of external sites. A membrane guards the incoming agents before allowing them to execute.

Recently, increasing attention has been devoted to express service contracts as behavioural (or session) types. These synthetise the essential aspects of the interaction behaviour of services, while allowing efficient static verification of properties of composed systems. Session types [24] have been exploited to formalize compatibility of components [37] and to describe adaptation of web services [16]. Security issues have been recently considered in terms of session types, e.g. in [13], which proves the decidability of type-checking in an extension of the π-calculus with session types and correspondence assertions [40].

Other works have proposed type-based methodologies to check security properties of distributed systems. For instance, Gordon and Jeffrey [22] use a type and effect system to prove authenticity properties of security protocols. Web service authentication has been recently modelled and analysed in [11, 12] through a process calculus enriched with cryptographic primitives.

The problem of discovering and composing Web Services by taking advantage of semantic information has been the subject of a considerable amount of research and development, [2, 17, 27, 29, 32, 36] to cite a few. The idea is to extend the primitives of service description languages with basic constructs for specifying properties of the published interface. We can distinguish between semantic-web descriptions [2, 29, 32, 36] in which service interfaces are annotated with parameter ontologies, and behavioural description [17, 27] in which the annotation details the ordering of service actions. A different solution to planning service composition has been proposed in [26], where the problem of composing services in order to achieve a given goal is expressed as a constraint satisfaction problem. Our approach extends and complements those based on behavioral descriptions, with an eye to security. Indeed, our methodology fully automates the process of discovering services and planning their composition in a secure way.

2 Planning Secure Service Compositions

To illustrate our approach, consider the scenario in the figure below. The boxes model services, distributed over a network. Each box encloses the service code, and is decorated with the location ℓ_i where the service is published.

Assume that the client at site ℓ_0 is a device with limited computational capabilities, wanting to execute some code downloaded from the network. To do that, the client issues two requests in sequence. The request labelled r_1 asks for a piece of mobile code (e.g. an applet), and it can be served by two code providers at ℓ_1 and ℓ_2. The request type $\tau \to (\tau \to \tau)$ means that, upon receiving a value of type τ (which can be an arbitrary base type, immaterial here) the invoked service replies with a function from τ to τ, with no security constraints.

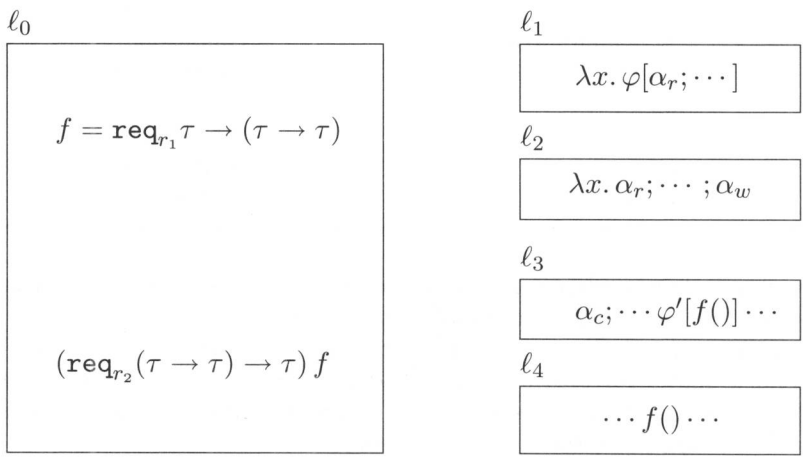

Fig. 1. One client (ℓ_0), two code providers (ℓ_1, ℓ_2), and two code executers (ℓ_3, ℓ_4)

The service at ℓ_1 returns a function that protects itself with a policy φ, permitting its use in certified sites only (modelled by the event α_c). Within the function body, the only security-relevant operation is a read α_r on the file system where the delivered code is run. The code provided by ℓ_2 first reads (α_r) some local data, and eventually writes them (α_w) back to ℓ_2.

Since ℓ_0 has a limited computational power, the code f obtained by the request r_1 is passed as a parameter to the service invoked by the request r_2. This request can be served by ℓ_3 and ℓ_4. The service at ℓ_3 is certified (α_c), and runs the provided code f under a "Chinese Wall" security policy φ', requiring that no data can be written (α_w) after reading them (α_r). The service at ℓ_4 is not certified, and it simply runs f.

2.1 Programming Model

Clients and services are modelled as expressions in a λ-calculus enriched with primitives for security and service requests. Security-relevant operations are rendered as side-effects in the calculus, and they are called *access events* (e.g. $\alpha_c, \alpha_r, \alpha_w$). A *security policy* is a regular property over a sequence η of access events, namely a *history*. A piece of code e framed within a policy φ (written $\varphi[e]$) must respect φ at each step of its execution. A *service request* has the form $\mathtt{req}_r\rho$. The label r uniquely identifies the request, while the *request type* ρ is the query pattern to be matched by the invoked service. For instance, the request type $\tau \xrightarrow{\varphi[\bullet]} \tau'$ matches services with functional type $\tau \to \tau'$, and whose behaviour respects the policy φ. The abstract syntax of services follows.

Syntax of Services

e, e'	::=	x	variable
		α	access event
		$\mathtt{if}\ b\ \mathtt{then}\ e\ \mathtt{else}\ e$	conditional
		$\lambda_z x.\, e$	named abstraction
		$e\, e'$	application
		$\varphi[e]$	safety framing
		$\mathtt{req}_r\rho$	service request
		$\mathtt{wait}\ \ell$	wait reply

The stand-alone evaluation of a service is much alike the call-by-value semantics of the λ-calculus; additionally, it enforces all the policies within their framings. More precisely, assume that, starting from the current history η, an expression e may evolve to e' and extend the history to η'. Then, a framing $\varphi[e]$ may evolve to $\varphi[e']$ if η' satisfies φ — otherwise the evaluation gets stuck. Eventually, values leave the scope of framings.

When a service is plugged into a network, a plan is used to resolve the requests therein, acting as an orchestrator. For brevity, we consider here only the case of *simple* plans, that have the following syntax: