Welf Löwe
Mario Südholt (Eds.)

LNCS 4089

Software Composition

5th International Workshop, SC 2006
Vienna, Austria, March 2006
Revised Selected Papers

Springer

Welf Löwe Mario Südholt (Eds.)

Software Composition

5th International Symposium, SC 2006
Vienna, Austria, March 25-26, 2006
Revised Papers

Springer

Volume Editors

Welf Löwe
Växjö University
School of Mathematics and Systems Engineering
Software Technology Group
351-95 Växjö, Sweden
E-mail: welf.lowe@msi.vxu.se

Mario Südholt
École des Mines de Nantes
Département Informatique
4, rue Alfred Kastler, 44307 Nantes Cedex 3, France
E-mail: Mario.Sudholt@emn.fr

Library of Congress Control Number: 2006930915

CR Subject Classification (1998): D.2, D.1.5, D.3, F.3

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-37657-7 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-37657-6 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11821946 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 4089

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

Preface

Research in software composition investigates models and techniques to build systems from predefined, pretested, reusable components instead of building them from scratch. In recent years, this idea has largely been adopted by industry. In the shape of service-oriented architecture, software composition has become an influential design paradigm, especially for the (re-)organization of the IT infrastructure of organizations. On the technical level, the standardization of Web services and other composition technologies has further matured.

Current research in software composition aims at (further) developing composition models and techniques. The aspect-oriented programming and design paradigm, for instance, has gained interest in the research community as a composition (support) model. Other current research questions concern the specification of component contracts, in particular making explicit its observable behavior, and methods of correct components composition. The International Symposium on Software Composition provides a premier forum for discussing these kinds of research questions and presenting original research results.

This LNCS volume contains the proceedings of the 5th International Symposium on Software Composition, which was held as a satellite event of the European Joint Conferences on Theory and Practice of Software (ETAPS) in Vienna, Austria, March, 25-26 2006. The symposium started with a keynote on "Semantically Enabled Service-Oriented Architectures" given by Dieter Fensel, Director of the Digital Research Institute. The main program consisted of presentations of research papers on software compositions. These proceedings contain the revised versions of the papers presented at SC 2006.

We selected 21 technical papers out of 60 submissions. Each paper went through a thorough revision processes and was reviewed by three to five reviewers followed by an electronic Program Committee discussion. We would like to thank the Program Committee members and the external reviewers for selecting a set of diverse and excellent papers and making SC 2006 a success.

We would like to express our gratitude to the European Network of Excellence on Aspect-Oriented Software Development (AOSD-Europe) and to the International Federation for Information Processing, Technical Committee on Software: Theory and Practice (IFIP, TC 2) for sponsoring this event. Finally, we would like to thank the organizers of ETAPS 2006 for hosting and providing an excellent organizational framework for SC 2006.

June 2006

Welf Löwe, Växjö University, Sweden
Mario Südholt, INRIA - École des Mines de Nantes, France
Program Co-chairs
SC 2006

Organization

Program Committee

Brian Barry	(Bedarra Research Labs, Canada)
Alexandre Bergel	(Trinity College Dublin, Ireland)
Judith Bishop	(University of Pretoria, South Africa)
Pierre Cointe	(Ecole des Mines de Nantes, France)
Vittorio Cortellessa	(University of L'Aquila, Italy)
Thierry Coupaye	(France Telecom, France)
Birgit Demuth	(Technische Universität Dresden, Germany)
Flavio De Paoli	(University of Milano Bicocca, Italy)
Dieter Fensel	(DERI Galway/Innsbruck, Ireland/Austria)
Volker Gruhn	(University of Leipzig, Germany)
Thomas Gschwind	(IBM Research, Switzerland)
Arno Jacobsen	(University of Toronto, Canada)
Mehdi Jazayeri	(University of Vienna, Austria)
Tom Henzinger	(EPF Lausanne, Switzerland)
Kung-Kiu Lau	(The University of Manchester, UK)
Karl Lieberherr	(Northeastern University, USA)
Welf Löwe (Co-chair)	(Växjö University, Sweden)
Mira Mezini	(Darmstadt University of Technology, Germany)
Claus Pahl	(Dublin City University, Ireland)
Arnd Poetzsch-Heffter	(University of Kaiserslautern, Germany)
Elke Pulvermüller	(Karlsruhe University of Technology, Germany)
Lionel Seinturier	(INRIA & LIP6, France)
Mario Südholt (Co-chair)	(INRIA & EMN, France)
Wim Vanderperren	(VU Brussels, Belgium)

Referees

U. Aßmann	E. Della Valle	S. Hu
O. Barais	M. D'Hondt	A. Jackson
D. Beyer	J. Feng	E. Kilgarriff
M. Book	D. Gao	L. Ling
F. Cabitza	V. Gasiunas	S. Loecher
O. Caron	M. Gawkowski	M. Loregian
A. Chakrabarti	F. Hartmann	A. Maurino
P.-C. David	T. Haselwanter	I. Ntalamagkas
B. De Fraine	M. Haupt	J. Oberleitner

J. Palm
M. Petrovic
N. Rauch
M. Reitz
N. Rivierre
D. Roman
R. Rouvoy
B. Sapkota
I. Savga
J. Schäfer
T. Schaefer
J. Scicluna
D. Suvee
T. Skotiniotis
F. M. Taweel
I. Toma
E. Tu
V. Ukis
Z. Wang
Z. Xu
M. Zaremba
St. Zschaler
C. Zhang
A. V. Zhdanova

Sponsoring Institutions

IFIP, Laxenburg, Austria
AOSD-Europe, European Network of Excellence in AOSD, Lancaster, UK

Lecture Notes in Computer Science

For information about Vols. 1–4028

please contact your bookseller or Springer

Vol. 4153: N. Zheng, X. Jiang, X. lan (Eds.), Advances in Machine Vision, Image Processing, and Pattern Analysis. XIII, 506 pages. 2006.

Vol. 4146: J.C. Rajapakse, L. Wong, R. Acharya (Eds.), Pattern Recognition in Bioinformatics. XIV, 186 pages. 2006. (Sublibrary LNBI).

Vol. 4144: T. Ball, R.B. Jones (Eds.), Computer Aided Verification. XV, 564 pages. 2006.

Vol. 4139: T. Salakoski, F. Ginter, S. Pyysalo, T. Pahikkala, Advances in Natural Language Processing. XVI, 771 pages. 2006. (Sublibrary LNAI).

Vol. 4138: X. Cheng, W. Li, T. Znati (Eds.), Wireless Algorithms, Systems, and Applications. XVI, 709 pages. 2006.

Vol. 4137: C. Baier, H. Hermanns (Eds.), CONCUR 2006 – Concurrency Theory. XIII, 525 pages. 2006.

Vol. 4133: J. Gratch, M. Young, R. Aylett, D. Ballin, P. Olivier (Eds.), Intelligent Virtual Agents. XIV, 472 pages. 2006. (Sublibrary LNAI).

Vol. 4130: U. Furbach, N. Shankar (Eds.), Automated Reasoning. XV, 680 pages. 2006. (Sublibrary LNAI).

Vol. 4129: D. McGookin, S. Brewster (Eds.), Haptic and Audio Interaction Design. XII, 167 pages. 2006.

Vol. 4127: E. Damiani, P. Liu (Eds.), Data and Applications Security XX. X, 319 pages. 2006.

Vol. 4121: A. Biere, C.P. Gomes (Eds.), Theory and Applications of Satisfiability Testing - SAT 2006. XII, 438 pages. 2006.

Vol. 4117: C. Dwork (Ed.), Advances in Cryptology - Crypto 2006. XIII, 621 pages. 2006.

Vol. 4115: D.-S. Huang, K. Li, G.W. Irwin (Eds.), Computational Intelligence and Bioinformatics, Part III. XXI, 803 pages. 2006. (Sublibrary LNBI).

Vol. 4114: D.-S. Huang, K. Li, G.W. Irwin (Eds.), Computational Intelligence, Part II. XXVII, 1337 pages. 2006. (Sublibrary LNAI).

Vol. 4113: D.-S. Huang, K. Li, G.W. Irwin (Eds.), Intelligent Computing, Part I. XXVII, 1331 pages. 2006.

Vol. 4112: D.Z. Chen, D. T. Lee (Eds.), Computing and Combinatorics. XIV, 528 pages. 2006.

Vol. 4111: F.S. de Boer, M.M. Bonsangue, S. Graf, W.-P. de Roever (Eds.), Formal Methods for Components and Objects. VIII, 447 pages. 2006.

Vol. 4109: D.-Y. Yeung, J.T. Kwok, A. Fred, F. Roli, D. de Ridder (Eds.), Structural, Syntactic, and Statistical Pattern Recognition. XXI, 939 pages. 2006.

Vol. 4108: J.M. Borwein, W.M. Farmer (Eds.), Mathematical Knowledge Management. VIII, 295 pages. 2006. (Sublibrary LNAI).

Vol. 4106: T.R. Roth-Berghofer, M.H. Göker, H. A. Güvenir (Eds.), Advances in Case-Based Reasoning. XIV, 566 pages. 2006. (Sublibrary LNAI).

Vol. 4104: T. Kunz, S.S. Ravi (Eds.), Ad-Hoc, Mobile, and Wireless Networks. XII, 474 pages. 2006.

Vol. 4099: Q. Yang, G. Webb (Eds.), PRICAI 2006: Trends in Artificial Intelligence. XXVIII, 1263 pages. 2006. (Sublibrary LNAI).

Vol. 4098: F. Pfenning (Ed.), Term Rewriting and Applications. XIII, 415 pages. 2006.

Vol. 4097: X. Zhou, O. Sokolsky, L. Yan, E.-S. Jung, Z. Shao, Y. Mu, D.C. Lee, D. Kim, Y.-S. Jeong, C.-Z. Xu (Eds.), Emerging Directions in Embedded and Ubiquitous Computing. XXVII, 1034 pages. 2006.

Vol. 4096: E. Sha, S.-K. Han, C.-Z. Xu, M.H. Kim, L.T. Yang, B. Xiao (Eds.), Embedded and Ubiquitous Computing. XXIV, 1170 pages. 2006.

Vol. 4094: O. H. Ibarra, H.-C. Yen (Eds.), Implementation and Application of Automata. XIII, 291 pages. 2006.

Vol. 4093: X. Li, O.R. Zaïane, Z. Li (Eds.), Advanced Data Mining and Applications. XXI, 1110 pages. 2006. (Sublibrary LNAI).

Vol. 4092: J. Lang, F. Lin, J. Wang (Eds.), Knowledge Science, Engineering and Management. XV, 664 pages. 2006. (Sublibrary LNAI).

Vol. 4091: G.-Z. Yang, T. Jiang, D. Shen, L. Gu, J. Yang (Eds.), Medical Imaging and Augmented Reality. XIII, 399 pages. 2006.

Vol. 4090: S. Spaccapietra, K. Aberer, P. Cudré-Mauroux (Eds.), Journal on Data Semantics VI. XI, 211 pages. 2006.

Vol. 4089: W. Löwe, M. Südholt (Eds.), Software Composition. X, 339 pages. 2006.

Vol. 4088: Z.-Z. Shi, R. Sadananda (Eds.), Agent Computing and Multi-Agent Systems. XVII, 827 pages. 2006. (Sublibrary LNAI).

Vol. 4085: J. Misra, T. Nipkow, E. Sekerinski (Eds.), FM 2006: Formal Methods. XV, 620 pages. 2006.

Vol. 4079: S. Etalle, M. Truszczyński (Eds.), Logic Programming. XIV, 474 pages. 2006.

Vol. 4077: M.-S. Kim, K. Shimada (Eds.), Advances in Geometric Modeling and Processing. XVI, 696 pages. 2006.

Vol. 4076: F. Hess, S. Pauli, M. Pohst (Eds.), Algorithmic Number Theory. X, 599 pages. 2006.

Vol. 4075: U. Leser, F. Naumann, B. Eckman (Eds.), Data Integration in the Life Sciences. XI, 298 pages. 2006. (Sublibrary LNBI).

Vol. 4074: M. Burmester, A. Yasinsac (Eds.), Secure Mobile Ad-hoc Networks and Sensors. X, 193 pages. 2006.

Vol. 4073: A. Butz, B. Fisher, A. Krüger, P. Olivier (Eds.), Smart Graphics. XI, 263 pages. 2006.

Vol. 4072: M. Harders, G. Székely (Eds.), Biomedical Simulation. XI, 216 pages. 2006.

Vol. 4071: H. Sundaram, M. Naphade, J.R. Smith, Y. Rui (Eds.), Image and Video Retrieval. XII, 547 pages. 2006.

Vol. 4070: C. Priami, X. Hu, Y. Pan, T.Y. Lin (Eds.), Transactions on Computational Systems Biology V. IX, 129 pages. 2006. (Sublibrary LNBI).

Vol. 4069: F.J. Perales, R.B. Fisher (Eds.), Articulated Motion and Deformable Objects. XV, 526 pages. 2006.

Vol. 4068: H. Schärfe, P. Hitzler, P. Øhrstrøm (Eds.), Conceptual Structures: Inspiration and Application. XI, 455 pages. 2006. (Sublibrary LNAI).

Vol. 4067: D. Thomas (Ed.), ECOOP 2006 – Object-Oriented Programming. XIV, 527 pages. 2006.

Vol. 4066: A. Rensink, J. Warmer (Eds.), Model Driven Architecture – Foundations and Applications. XII, 392 pages. 2006.

Vol. 4065: P. Perner (Ed.), Advances in Data Mining. XI, 592 pages. 2006. (Sublibrary LNAI).

Vol. 4064: R. Büschkes, P. Laskov (Eds.), Detection of Intrusions and Malware & Vulnerability Assessment. X, 195 pages. 2006.

Vol. 4063: I. Gorton, G.T. Heineman, I. Crnkovic, H.W. Schmidt, J.A. Stafford, C.A. Szyperski, K. Wallnau (Eds.), Component-Based Software Engineering. XI, 394 pages. 2006.

Vol. 4062: G. Wang, J.F. Peters, A. Skowron, Y. Yao (Eds.), Rough Sets and Knowledge Technology. XX, 810 pages. 2006. (Sublibrary LNAI).

Vol. 4061: K. Miesenberger, J. Klaus, W. Zagler, A. Karshmer (Eds.), Computers Helping People with Special Needs. XXIX, 1356 pages. 2006.

Vol. 4060: K. Futatsugi, J.-P. Jouannaud, J. Meseguer (Eds.), Algebra, Meaning, and Computation. XXXVIII, 643 pages. 2006.

Vol. 4059: L. Arge, R. Freivalds (Eds.), Algorithm Theory – SWAT 2006. XII, 436 pages. 2006.

Vol. 4058: L.M. Batten, R. Safavi-Naini (Eds.), Information Security and Privacy. XII, 446 pages. 2006.

Vol. 4057: J.P.W. Pluim, B. Likar, F.A. Gerritsen (Eds.), Biomedical Image Registration. XII, 324 pages. 2006.

Vol. 4056: P. Flocchini, L. Gąsieniec (Eds.), Structural Information and Communication Complexity. X, 357 pages. 2006.

Vol. 4055: J. Lee, J. Shim, S.-g. Lee, C. Bussler, S. Shim (Eds.), Data Engineering Issues in E-Commerce and Services. IX, 290 pages. 2006.

Vol. 4054: A. Horváth, M. Telek (Eds.), Formal Methods and Stochastic Models for Performance Evaluation. VIII, 239 pages. 2006.

Vol. 4053: M. Ikeda, K.D. Ashley, T.-W. Chan (Eds.), Intelligent Tutoring Systems. XXVI, 821 pages. 2006.

Vol. 4052: M. Bugliesi, B. Preneel, V. Sassone, I. Wegener (Eds.), Automata, Languages and Programming, Part II. XXIV, 603 pages. 2006.

Vol. 4051: M. Bugliesi, B. Preneel, V. Sassone, I. Wegener (Eds.), Automata, Languages and Programming, Part I. XXIII, 729 pages. 2006.

Vol. 4049: S. Parsons, N. Maudet, P. Moraitis, I. Rahwan (Eds.), Argumentation in Multi-Agent Systems. XIV, 313 pages. 2006. (Sublibrary LNAI).

Vol. 4048: L. Goble, J.-J.C.. Meyer (Eds.), Deontic Logic and Artificial Normative Systems. X, 273 pages. 2006. (Sublibrary LNAI).

Vol. 4047: M. Robshaw (Ed.), Fast Software Encryption. XI, 434 pages. 2006.

Vol. 4046: S.M. Astley, M. Brady, C. Rose, R. Zwiggelaar (Eds.), Digital Mammography. XVI, 654 pages. 2006.

Vol. 4045: D. Barker-Plummer, R. Cox, N. Swoboda (Eds.), Diagrammatic Representation and Inference. XII, 301 pages. 2006. (Sublibrary LNAI).

Vol. 4044: P. Abrahamsson, M. Marchesi, G. Succi (Eds.), Extreme Programming and Agile Processes in Software Engineering. XII, 230 pages. 2006.

Vol. 4043: A.S. Atzeni, A. Lioy (Eds.), Public Key Infrastructure. XI, 261 pages. 2006.

Vol. 4042: D. Bell, J. Hong (Eds.), Flexible and Efficient Information Handling. XVI, 296 pages. 2006.

Vol. 4041: S.-W. Cheng, C.K. Poon (Eds.), Algorithmic Aspects in Information and Management. XI, 395 pages. 2006.

Vol. 4040: R. Reulke, U. Eckardt, B. Flach, U. Knauer, K. Polthier (Eds.), Combinatorial Image Analysis. XII, 482 pages. 2006.

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. XVII, 474 pages. 2006.

Vol. 4036: O. H. Ibarra, Z. Dang (Eds.), Developments in Language Theory. XII, 456 pages. 2006.

Vol. 4035: T. Nishita, Q. Peng, H.-P. Seidel (Eds.), Advances in Computer Graphics. XX, 771 pages. 2006.

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

Vol. 4033: B. Stiller, P. Reichl, B. Tuffin (Eds.), Performability Has its Price. X, 103 pages. 2006.

Vol. 4032: O. Etzion, T. Kuflik, A. Motro (Eds.), Next Generation Information Technologies and Systems. XIII, 365 pages. 2006.

Vol. 4031: M. Ali, R. Dapoigny (Eds.), Advances in Applied Artificial Intelligence. XXIII, 1353 pages. 2006. (Sublibrary LNAI).

Vol. 4029: L. Rutkowski, R. Tadeusiewicz, L.A. Zadeh, J.M. Zurada (Eds.), Artificial Intelligence and Soft Computing – ICAISC 2006. XXI, 1235 pages. 2006. (Sublibrary LNAI).

Table of Contents

Automatic Checking of Component Protocols in Component-Based Systems

Wolf Zimmermann[1] and Michael Schaarschmidt[2]

[1] Martin-Luther Universität Halle-Wittenberg, Institut für Informatik,
06099 Halle/Saale, Germany
zimmer@informatik.uni-halle.de
[2] Martin-Luther Universität Halle-Wittenberg, Rechenzentrum,
06099 Halle/Saale, Germany
michael.schaarschmidt@urz.uni-halle.de

Abstract. We statically check whether each component in a component-based system is used according to its protocol and provide counterexamples if such a check fails. The protocol is given by a finite state machine specifying legal sequences of procedure calls of the interface of a component. The main contribution is that we can deal with call-backs without any restrictions. We achieved this by using context-free grammars instead of finite state machines to describe the use of components.

1 Introduction

The construction of component-based systems became increasingly important in software construction. However, software architects have to deal with new problems stemming from component-based system architectures. An important issue is whether a component is correctly used. Usually components implement one or more interfaces specifying the services they offer. For the purpose of this paper, a *service* is simply a procedure or function signature. However just the knowledge of services does not provide sufficient information for the construction of systems. Often the source code of a component is not available after its deployment or even not physically available as e.g. Web Services. However, for a component industry the unavailability of source code is essential – Web Services may even be offered on a pay-per-use basis.

A major problem for construction of component-based systems is to check whether the components can be composed and possibly provide own components to adapt them. A failure to use a component correctly might cause a system abortion while executing the system – this might happen even after the system is delivered to the customer. In this context abortion means that a system stops with an uncaught exception internal to a component (e.g. dereferencing of null reference, illegal array accesses, division by zero etc.). Since the source code of components is often unavailable, other approaches are necessary to check whether components are used in such a way that the system does not abort. Our goal is to provide a mechanizable approach for checking statically component-based systems for abortion freeness on an almost black-box basis.

W. Löwe and M. Südholt (Eds.): SC 2006, LNCS 4089, pp. 1–17, 2006.

Our approach currently restricts the architecture of component-based systems to sequential systems and to one client using services of other components. However any component may use services of other components or even of the client. In particular we do not exclude call-backs. We assume that each component implements one or more interfaces and each interface I specifies services as a set of procedure signatures Σ_I. The *services* Σ_C of a component C is the union of the interfaces implemented by the component. Informally, a *protocol* of a component is a set of sequences $L_C \subseteq \Sigma_C^*$. The aim of protocols is to guarantee certain properties, e.g. that the component doesn't abort if its services are called according to a sequence in L_C. A component C might call other services specified as interfaces used by a component. The *profile* of a component C specifies for each interface I required by C the set of sequences $P_C(I) \subseteq \Sigma_I^*$ of services possibly being called by C. A component-based system $\mathcal{S}$ is a multi-set of components (i.e. there might be multiple copies of a component called *instances*) where each interface used by a component is instantiated with an instance of a component. The *use of an instance* c of a component C in a component-based system $\mathcal{S}$ is the set of possible sequences of services $U_c \subseteq \Sigma_C^*$ that are called to c during execution of $\mathcal{S}$. The use of an instance c of a component C *conforms* to its protocol iff $U_c \subseteq L_C$, i.e. any sequence of services called to c agrees with the protocol of C. Therefore, if the conformity check succeeds for each instance in a component-based system $\mathcal{S}$ and each component of $\mathcal{S}$ is correctly implemented then the abortion-freeness of $\mathcal{S}$ is guaranteed. We assume that each component contains in its deployment description its protocol and its profile.

Many approaches (e.g. [13,17,18,22]) use finite state machines (short: FSM) A_P and A_R to specify protocols $L(A_P)$ and profiles $L(A_R)$ for each interface of a component where $L(A_P)$ and $L(A_R)$ are the languages accepted by A_P and A_R, respectively. Since connectors connect profiles for an interface of one component with an interface of another component it is checked whether $L(A_R) \subseteq L(A_P)$ and counter-examples are provided in the case such a check fails. The idea behind these local checks is that protocol conformance checks can be executed incrementally. It implicitly assumes that any checked connection cannot be invalidated as long as the protocols of the component providing the profile are satisfied. In this paper, we show that these approaches have several drawbacks: First, local checks cannot be applied if the interfaces of a component cannot be used independently. Second, it cannot be applied if the component system contains recursive call-backs.

Other works use model-checking approaches [8,9,7,11,3,5] to prove that programs satisfy certain properties. They use context-free model checking because finite state machine models are not an adequate abstraction if the program may contain recursive procedures. However, these works assume that the whole program is completely available.

Our method combines and generalizes these approaches in order to allow dependencies between different interfaces of a component and arbitrary recursive call-backs. FSMs are used for describing protocols. In contrast to the above

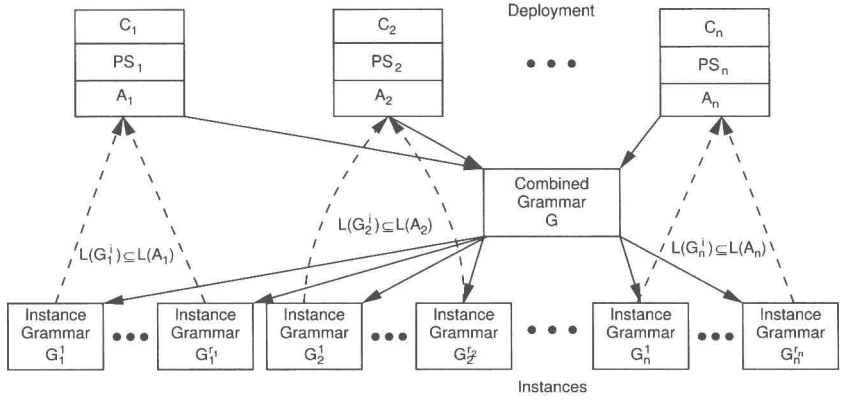

Fig. 1. An Approach to Conformance Checking

works, a single FSM A_C is used for the whole component C. Hence, interaction of procedure calls to different interfaces of C are taken into account. Instead of FSMs for describing uses of components our approach uses context-free grammar (short: CFG) G_C for this purpose. Thus, for each instance c of a component C it is checked whether $L(G_c) \subseteq L(A_C)$. It is a well-known result from the theory of formal languages that this test is algorithmically decidable. We show how counterexamples can be provided if such a check fails. From the global system and the profiles of each component the use of components is derived. However, a profile of a component C cannot be described itself as a context-free grammar since only the use of interfaces is known but not how these are instantiated. Therefore, we generalize context-free grammars by parameterizing non-terminal and terminal symbols with the interfaces. The obtained structure is called a *parameterized context-free scheme* (pCFS). These pCFSs can be mechanically computed from the source code of the components. The pCFSs are used in a component-based system $\mathcal{S}$ to compute a context free grammar specifying all sequences of calls to all instances of the components by instantiating the interfaces of the pCFS analogous to the corresponding instances in $\mathcal{S}$. Context-free grammars for uses of instances of components in a component-based system can now be derived by projection. Hence, it is now possible to check for each instance c of component C of $\mathcal{S}$ whether $L(G_c) \subseteq L(A_C)$, i.e. whether to use of c conforms to the protocol. Fig. 1 illustrates the summary of our approach. The paper is organized as follows: Section 2 demonstrates the limitations of local checks and the use of FSMs for profiles. Section 3 summarizes how to check $L(G) \subseteq L(A)$ for CFGs G and FSMs A and shows how counterexamples are provided. Section 4 introduces parameterized context-free schemes. Section 5 shows their use in specifying profiles of components and how they can be generated from source text. Finally, Section 6 shows how CFGs specifying the use of a component are generated from the profiles. A short appendix introduces some of the notations from formal languages used in this paper.

2 Limitations of Local Checks Based on FSMs

Local protocol checking approaches (e.g. [18,20,13]) check independently each connection in a component-based system. They usually assume a protocol for each interface and assume that they can be independently used. I.e. instances of a component C can accept all interleavings of all calling sequences by the protocols of its interfaces. Then they deduce profiles for the interfaces required by C. Hence, it is possible to check whether a profile U for an interface conforms to its protocol P, i.e. whether $U \subseteq P$. For these checks, it is often assumed that the profile U also is a regular language. Thus, protocol conformance can be decided. However, in practice it often happens that components cannot accept arbitrary interleavings of the calling sequences to its protocol. Thus, more sophisticated approaches introduce coordination components (sometimes also called connectors) that accept arbitrary interleavings and the other components only have one interface. Therefore a component has a single protocol. In this section, we show that even if each component has one interface and if each connection is succesfully checked, the absence of global protocol errors is not guaranteed. The main reason for these violations are recursive call-backs. Thus, for the same reasons as in the works of software model checking [8,9,7,11,3,5], CFGs are more adequate than FSMs to describe the use of components.

Our examples are denoted similar to Java. The main difference is that classes are components and we do not inherit from components. Procedures and functions can only have parameters whose types are interfaces or basic types (for simplicity, we only use here the type *int*). Any procedure or function that is not defined by an interface of a component is *internal* to that component. There is exactly one component, the *client*, containing a parameterless procedure *main* which is executed upon on system start. The client has parameters that represent interfaces to be instantiated with components upon composition time. Thus, all instances of components in a component based system are known upon composition time. Note that all instances of a component can be referenced by a name. Procedures allow to pass by reference instances of components. Values of basic types are passed by value. This model is similar to commercial component systems such as COM, EJB, CORBA except that all components are known upon composition time. Dynamic instances of components are possible. The operation $new(x)$ computes a new instance of the component refered to by variable x. In this paper, we assume for simplicity that all services of components are procedures. The parameters of the client can only be used in *main*. The identifier *this* denotes the instance of the component currently being executed.

Example 1. Consider the component system in Fig. 2. c is an instance of C_2. The component system starts its execution by executing *main*. Suppose we read 2, i.e. $i = 2$. Then, the body of the loop will be executed and it calls $c.a(2, this)$. Thus, when executing this call on c it is $n = 2$ and x refers to the client. Since the condition becomes true, the call $x.b(1, this)$ is being executed. Since x refers to the client, this is a call-back and the execution of b on the client starts with $k = 1$ and z referring to c. After the first assignment it holds $n = 1$ which also

```
client C1[J c] implements I {
    int n = 0;
    void b(int k, J z){
        n = (n + 1)%2; n = 1/n;
        z.a(k, this);
    }
    void d(){ n = 1/n − 1; }
    void main(){
        int i;
        read(i);
        while(i > 0){ c.a(i, this); i − −; }
    }
}

interface I {
    void b(int, J);
    void d(J);
}
```

```
component C2 implements J {
    void a(int n, I x){
        if(n > 0) { x.b(n − 1, this); x.d(); }
    }
}

interface J {
    void a(int, I);
}
```

Composition: $C_1[C_2]$

Fig. 2. A Component-Based System

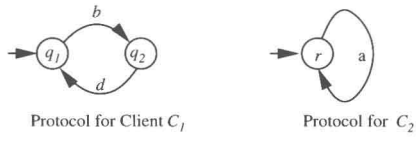

Fig. 3. Protocols for the Client and Component in Fig. 2

holds after the second assignment. Thus, the call $z.a(1, this)$ is executed. Note that this is a *recursive* call since z refers to c and the first call of a on c is not yet completed. In this second call it is $n = 1$ and x refers to the client. Thus the condition becomes true and the call $x.b(0, this)$ is being executed. This again is a recursive call since x refers to the client and the first call of b on the client is not yet completed. After the execution of the first statement it holds $n = 0$. Hence, the second statements performs a division by 0 and therefore the system aborts.

Fig. 3 shows the protocol of the components. Note that a second execution of b and a second execution of d on the client lead to a division by zero. The client requires that b and d must be called alternating and b is called first – if at all. Otherwise divisions by zero are executed. Apparently, this protocol is violated by the system.

The following example demonstrates that recursive call-backs are the reason for protocol violations:

Example 2. According to the clients protocol, the profile for c is $L_c = \{a^n | n \in \mathbb{N}\}$ and the profile for z is also $L_z = \{a^n | n \in \mathbb{N}\}$. According to component C_2's protocol, the profile for x is $L_x = \{(bd)^n | n \in \mathbb{N}\}$.

After composition, z always refers to c and x always refers to the client. Such information could e.g. be derived from a points-to analysis. Thus there are two profiles for calling sequences to c. Even an arbitrary interleaving of L_z and L_c shows that $U_c = \{a^n | n \in \mathbb{N}\}$ is the set of all calling sequences to instance c of

component C_2. Since these sequences are accepted, the use of c conforms to the protocol of C. Consider now the client. Since x is the only variable referring to the client, the use of the client is $U = \{(bd)^n | n \in \mathbb{N}\}$. Hence, the local protocol checking approach also would decide that the use of the client conforms to its protocol which is wrong according to the scenario in Example 1.

The checking approach in Example 2 considers individually each component. If there wouldn't introduced recursive procedure calls due to call-backs the above arguments would be completely legal. The individual protocol conformance checking doesn't work because these recursive calls lead to use of components that cannot be detected from one component alone. Many works of protocol checking are aware of this problem and exclude therefore recursive call-backs. In fact if a is recursively called every call b can be viewed as an open bracket that is closed by a call d. Therefore the set of sequences describing the use of the client is the Dyck-Language over the pair of brackets b and d. It is generated by the CFG $G = (\{b, d\}, \{Z\}, \{Z ::= ZZ | cZd | \varepsilon\}, Z)$. It is a well-known result from the theory of formal languages that Dyck-Languages are not regular languages and therefore no FSM exists that accepts Dyck-languages. Thus, the use of components cannot be specified using FSMs. The next section shows that even in the case that the use of components is described by CFGs, model checking of protocol conformance is possible.

3 Model Checking with CFGs

We present here the standard algorithm for checking $L(G) \subseteq L(A)$ for a CFG $G = (T, N, P, Z)$ and a FSM $A = (T, Q, R, q_0, F)$. Furthermore, we show how it can be used to provide counterexamples if $L(G) \not\subseteq L(A)$. The basic idea is instead of checking $L(G) \subseteq L(A)$ to check the equivalent condition $L(G) \cap (T^* \setminus L(A)) = \emptyset$. Any word $w \in L(G) \cap (T^* \setminus L(A))$ is a counterexample of the check. In the context of the paper, it provides a sequence of procedure calls to a component that violate its protocol. The FSM $A' = (T, Q, R, q_0, Q \setminus F)$ accepts $T^* \setminus L(A)$. Hence, we check whether $L(G) \cap L(A') = \emptyset$. It is known that the intersection of a context-free language with a regular language is context-free, and that it is decidable for context-free grammars G whether $L(G) = \emptyset$. Our model checker therefore has the following steps: First, a CFG G' such that $L(G') = L(G) \cap L(A')$ is constructed. Then, it is checked whether $L(G') = \emptyset$. If it turns out that $L(G') \neq \emptyset$, a counterexample $w \in L(G')$ is produced.

Step 1: First the CFG $G = (T, N, P, Z)$ is transformed into an equivalent grammar in *extended Chomsky Normal Form* (short: eCNF) $G_1 = (T, N_1, P_1, Z_1)$, i.e., each production has one of the forms[1] $A ::= BC$, $A ::= B$, or $A ::= t$ with $A, B, C \in N$ and $t \in T$. If $\varepsilon \in L(G)$ then $Z_1 ::= \varepsilon \in P_1$. Second, a CFG $G' = (T, N', P', Z')$ is computed such that $L(G') = L(G_1) \cap L(A')$. Define the size of a CFG $G = (T, N, P, Z)$ as $|G| \triangleq |P| + \sum_{l::=r \in P} |r|$.

[1] Chomsky Normal form also forbids chain productions $A ::= B$.