Jan Bosch
Charles Krueger (Eds.)

LNCS 3107

Software Reuse: Methods, Techniques, and Tools

8th International Conference, ICSR 2004
Madrid, Spain, July 2004
Proceedings

Springer

Jan Bosch Charles Krueger (Eds.)

Software Reuse: Methods, Techniques, and Tools

8th International Conference, ICSR 2004
Madrid, Spain, July 5-9, 2004
Proceedings

Volume Editors

Jan Bosch
University of Groningen, Department of Computing Science
P.O. Box 800, 9700 AV, Groningen, Netherlands
E-mail: Jan.Bosch@cs.rug.nl

Charles Krueger
BigLever Software
10500 Laurel Hill Cove, Austin, TX, 78730, USA
E-mail: ckrueger@biglever.com

Library of Congress Control Number: Applied for

CR Subject Classification (1998): D.2, K.6, D.1, J.1

ISSN 0302-9743
ISBN 3-540-22335-5 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by PTP-Berlin, Protago-TeX-Production GmbH
Printed on acid-free paper SPIN: 11015529 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 3107

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

Preface

After three decades of research and practice, reuse of existing software artefacts remains the most promising approach to decreasing effort for software development and evolution, increasing quality of software artefacts and decreasing time to market of software products. Over time, we have seen impressive improvements, in extra-organizational reuse, e.g. COTS, as well as in intra-organizational reuse, e.g. software product families.

Despite the successes that we, as a community, have achieved, several challenges remain to be addressed. The theme for this eighth meeting of the premier international conference on software reuse is the management of software variability for reusable software. All reusable software operates in multiple contexts and has to accommodate the differences between these contexts through variation. In modern software, the number of variation points may range in the thousands with an even larger number of dependencies between these points. Topics addressing the theme include the representation, design, assessment and evolution of software variability.

The proceedings that you are holding as you read this report on the current state-of-the-art in software reuse. Topics covered in the proceedings include software variability, testing of reusable software artefacts, feature modeling, aspect-oriented software development, composition of components and services, model-based approaches and several other aspects of software reuse.

May 2004

Jan Bosch
Charles Krueger

Organizing Committee

General Chair	Kyo C. Kang, Pohang University of Science and Technology, Korea
Program Co-chairs	Jan Bosch, University of Groningen, The Netherlands Charles Krueger, BigLever Software, Inc., U.S.A.
Tutorial Chair	Sergio Bandinelli, European Software Institute, Spain
Workshop Chair	Klaus Schmid, Fraunhofer IESE, Germany
Doctoral Student Session Chair	Sholom Cohen, Software Engineering Institute, USA
Local Chair	Juan Llorens, Universidad Carlos III de Madrid, Spain
Finance and Registration Chair	Ernesto Guerrieri, GTECH Corporation, USA
Corporate Chair	Chuck Lillie, The University of North Carolina at Pembroke, USA
Publicity Chair	Jaejoon Lee, Pohang University of Science and Technology, Korea
Web Chair	Eelke Folmer, University of Groningen, The Netherlands
Advisory Committee	Ted Biggerstaff, Softwaregenerators.com, USA John Favaro, independent consultant, Italy Bill Frakes, Virginia Tech, USA

Sponsors

ISASE
Korea IT Industry Promotion Agency
SRA Key Technology Laboratory, Inc.
Consulenza Informatica, Italy
SRA-KTL, Japan
KIPA, Korea

Program Committee

Omar Alonso	Oracle, USA
Mikio Aoyama	Nanzan University, Japan
Sidney Bailin	Knowledge Evolution, USA
Sergio Bandinelli	European Software Institute, Spain
Len Bass	Software Engineering Institute, USA
Ira Baxter	SemanticDesigns, USA
Luigi Benedicenti	University of Regina, Canada
Per Olof Bengtsson	Ericsson, Sweden
Cornelia Boldyreff	University of Durham, UK
Paul Clements	Software Engineering Institute, USA
Sholom Cohen	Software Engineering Institute, USA
Jacob Cybulski	University of Melbourne, Australia
Krzysztof Czarnecki	University of Waterloo, Canada
Juan Carlos Duenas	Universidad Politécnica de Madrid, Spain
Philip Fong	University of Regina, Canada
Cristina Gacek	University of Newcastle upon Tyne, UK
Birget Geppert	Avaya, USA
Hassan Gomaa	George Mason University, USA
Stan Jarzabek	National University of Singapore, Singapore
Merijn de Jonge	Universiteit Utrecht, The Netherlands
Itoh Kiyoshi	Sophia University, Japan
Peter Knauber	Mannheim University of Applied Sciences, Germany
Kwanwoo Lee	Hansung University, Korea
Julio Cesar Leite	PUC-RIO, Brazil
Mike Mannion	Glasgow Caledonian University, UK
Michele Marchesi	University of Cagliari, Italy
Ali Mili	New Jersey Institute of Technology, USA
Roland Mittermeir	University of Klagenfurt, Austria
Maurizio Morisio	Politecnico di Torino, Italy
Hausi Muller	University of Victoria, Canada
David Mussser	Rensselaer Polytechnic Institute, USA
Dirk Muthig	Fraunhofer IESE, Germany

Jim Neighbors	Bayfront Technologies, USA
Jim Ning	Accenture, USA
Henk Obbink	Philips Research, The Netherlands
Sooyong Park	Sogang University, Korea
Witold Pedrycz	University of Alberta, Canada
John Penix	NASA, USA
Jeff Poulin	LockheedMartin, USA
Wolfgang Pree	University of Salzburg, Austria
Rubin Prieto-Diaz	James Madison University, USA
Alexander Romanovsky	University of Newcastle upon Tyne, UK
William Scherlis	Carnegie Mellon University, USA
Klaus Schmid	Fraunhofer IESE, Germany
Erwin Schoitsch	Austrian Research Centers, Austria
Murali Sitaraman	Clemson, USA
Douglas Smith	Kestrel Institute, USA
Giancarlo Succi	Free University of Bozen, Italy
Joost Visser	Universidade do Minho, Portugal
Steven Wartik	IDA, USA
Claudia Werner	University of Rio de Janeiro, Brazil
Gabi Zodik	IBM, Israel
Greg Kulczycki	Virginia Tech, USA

Lecture Notes in Computer Science

For information about Vols. 1–3021

please contact your bookseller or Springer-Verlag

Vol. 3125: D. Kozen (Ed.), Mathematics of Program Construction. X, 401 pages. 2004.

Vol. 3123: A. Belz, R. Evans, P. Piwek (Eds.), Natural Language Generation. X, 219 pages. 2004. (Subseries LNAI).

Vol. 3120: J. Shawe-Taylor, Y. Singer (Eds.), Learning Theory. X, 648 pages. 2004. (Subseries LNAI).

Vol. 3118: K. Miesenberger, J. Klaus, W. Zagler, D. Burger (Eds.), Computer Helping People with Special Needs. XXIII, 1191 pages. 2004.

Vol. 3116: C. Rattray, S. Maharaj, C. Shankland (Eds.), Algebraic Methodology and Software Technology. XI, 569 pages. 2004.

Vol. 3114: R. Alur, D.A. Peled (Eds.), Computer Aided Verification. XII, 536 pages. 2004.

Vol. 3113: J. Karhumäki, H. Maurer, G. Paun, G. Rozenberg (Eds.), Theory Is Forever. X, 283 pages. 2004.

Vol. 3112: H. Williams, L. MacKinnon (Eds.), New Horizons in Information Management. XII, 265 pages. 2004.

Vol. 3111: T. Hagerup, J. Katajainen (Eds.), Algorithm Theory - SWAT 2004. XI, 506 pages. 2004.

Vol. 3109: S.C. Sahinalp, S. Muthukrishnan, U. Dogrusoz (Eds.), Combinatorial Pattern Matching. XII, 486 pages. 2004.

Vol. 3107: J. Bosch, C. Krueger (Eds.), Software Reuse: Methods, Techniques and Tools. XI, 339 pages. 2004.

Vol. 3105: S. Göbel, U. Spierling, A. Hoffmann, I. Iurgel, O. Schneider, J. Dechau, A. Feix (Eds.), Technologies for Interactive Digital Storytelling and Entertainment. XVI, 304 pages. 2004.

Vol. 3104: R. Kralovic, O. Sykora (Eds.), Structural Information and Communication Complexity. X, 303 pages. 2004.

Vol. 3103: K. Deb (Ed.), Genetic and Evolutionary Computation - GECCO 2004. XLIX, 1439 pages. 2004.

Vol. 3102: K. Deb (Ed.), Genetic and Evolutionary Computation - GECCO 2004. L, 1445 pages. 2004.

Vol. 3101: M. Masoodian, S. Jones, B. Rogers (Eds.), Computer Human Interaction. XIV, 694 pages. 2004.

Vol. 3099: J. Cortadella, W. Reisig (Eds.), Applications and Theory of Petri Nets 2004. XI, 505 pages. 2004.

Vol. 3098: J. Desel, W. Reisig, G. Rozenberg (Eds.), Lectures on Concurrency and Petri Nets. VIII, 849 pages. 2004.

Vol. 3097: D. Basin, M. Rusinowitch (Eds.), Automated Reasoning. XII, 493 pages. 2004. (Subseries LNAI).

Vol. 3096: G. Melnik, H. Holz (Eds.), Advances in Learning Software Organizations. X, 173 pages. 2004.

Vol. 3094: A. Nürnberger, M. Detyniecki (Eds.), Adaptive Multimedia Retrieval. VIII, 229 pages. 2004.

Vol. 3093: S.K. Katsikas, S. Gritzalis, J. Lopez (Eds.), Public Key Infrastructure. XIII, 380 pages. 2004.

Vol. 3092: J. Eckstein, H. Baumeister (Eds.), Extreme Programming and Agile Processes in Software Engineering. XVI, 358 pages. 2004.

Vol. 3091: V. van Oostrom (Ed.), Rewriting Techniques and Applications. X, 313 pages. 2004.

Vol. 3089: M. Jakobsson, M. Yung, J. Zhou (Eds.), Applied Cryptography and Network Security. XIV, 510 pages. 2004.

Vol. 3086: M. Odersky (Ed.), ECOOP 2004 – Object-Oriented Programming. XIII, 611 pages. 2004.

Vol. 3085: S. Berardi, M. Coppo, F. Damiani (Eds.), Types for Proofs and Programs. X, 409 pages. 2004.

Vol. 3084: A. Persson, J. Stirna (Eds.), Advanced Information Systems Engineering. XIV, 596 pages. 2004.

Vol. 3083: W. Emmerich, A.L. Wolf (Eds.), Component Deployment. X, 249 pages. 2004.

Vol. 3080: J. Desel, B. Pernici, M. Weske (Eds.), Business Process Management. X, 307 pages. 2004.

Vol. 3079: Z. Mammeri, P. Lorenz (Eds.), High Speed Networks and Multimedia Communications. XVIII, 1103 pages. 2004.

Vol. 3078: S. Cotin, D.N. Metaxas (Eds.), Medical Simulation. XVI, 296 pages. 2004.

Vol. 3077: F. Roli, J. Kittler, T. Windeatt (Eds.), Multiple Classifier Systems. XII, 386 pages. 2004.

Vol. 3076: D. Buell (Ed.), Algorithmic Number Theory. XI, 451 pages. 2004.

Vol. 3074: B. Kuijpers, P. Revesz (Eds.), Constraint Databases and Applications. XII, 181 pages. 2004.

Vol. 3073: H. Chen, R. Moore, D.D. Zeng, J. Leavitt (Eds.), Intelligence and Security Informatics. XV, 536 pages. 2004.

Vol. 3072: D. Zhang, A.K. Jain (Eds.), Biometric Authentication. XVII, 800 pages. 2004.

Vol. 3070: L. Rutkowski, J. Siekmann, R. Tadeusiewicz, L.A. Zadeh (Eds.), Artificial Intelligence and Soft Computing - ICAISC 2004. XXV, 1208 pages. 2004. (Subseries LNAI).

Vol. 3068: E. André, L. Dybkjær, W. Minker, P. Heisterkamp (Eds.), Affective Dialogue Systems. XII, 324 pages. 2004. (Subseries LNAI).

Vol. 3067: M. Dastani, J. Dix, A. El Fallah-Seghrouchni (Eds.), Programming Multi-Agent Systems. X, 221 pages. 2004. (Subseries LNAI).

Vol. 3066: S. Tsumoto, R. Slowiński, J. Komorowski, J.W. Grzymala-Busse (Eds.), Rough Sets and Current Trends in Computing. XX, 853 pages. 2004. (Subseries LNAI).

Vol. 3065: A. Lomuscio, D. Nute (Eds.), Deontic Logic in Computer Science. X, 275 pages. 2004. (Subseries LNAI).

Vol. 3064: D. Bienstock, G. Nemhauser (Eds.), Integer Programming and Combinatorial Optimization. XI, 445 pages. 2004.

Vol. 3063: A. Llamosí, A. Strohmeier (Eds.), Reliable Software Technologies - Ada-Europe 2004. XIII, 333 pages. 2004.

Vol. 3062: J.L. Pfaltz, M. Nagl, B. Böhlen (Eds.), Applications of Graph Transformations with Industrial Relevance. XV, 500 pages. 2004.

Vol. 3061: F.F. Ramas, H. Unger, V. Larios (Eds.), Advanced Distributed Systems. VIII, 285 pages. 2004.

Vol. 3060: A.Y. Tawfik, S.D. Goodwin (Eds.), Advances in Artificial Intelligence. XIII, 582 pages. 2004. (Subseries LNAI).

Vol. 3059: C.C. Ribeiro, S.L. Martins (Eds.), Experimental and Efficient Algorithms. X, 586 pages. 2004.

Vol. 3058: N. Sebe, M.S. Lew, T.S. Huang (Eds.), Computer Vision in Human-Computer Interaction. X, 233 pages. 2004.

Vol. 3057: B. Jayaraman (Ed.), Practical Aspects of Declarative Languages. VIII, 255 pages. 2004.

Vol. 3056: H. Dai, R. Srikant, C. Zhang (Eds.), Advances in Knowledge Discovery and Data Mining. XIX, 713 pages. 2004. (Subseries LNAI).

Vol. 3055: H. Christiansen, M.-S. Hacid, T. Andreasen, H.L. Larsen (Eds.), Flexible Query Answering Systems. X, 500 pages. 2004. (Subseries LNAI).

Vol. 3054: I. Crnkovic, J.A. Stafford, H.W. Schmidt, K. Wallnau (Eds.), Component-Based Software Engineering. XI, 311 pages. 2004.

Vol. 3053: C. Bussler, J. Davies, D. Fensel, R. Studer (Eds.), The Semantic Web: Research and Applications. XIII, 490 pages. 2004.

Vol. 3052: W. Zimmermann, B. Thalheim (Eds.), Abstract State Machines 2004. Advances in Theory and Practice. XII, 235 pages. 2004.

Vol. 3051: R. Berghammer, B. Möller, G. Struth (Eds.), Relational and Kleene-Algebraic Methods in Computer Science. X, 279 pages. 2004.

Vol. 3050: J. Domingo-Ferrer, V. Torra (Eds.), Privacy in Statistical Databases. IX, 367 pages. 2004.

Vol. 3049: M. Bruynooghe, K.-K. Lau (Eds.), Program Development in Computational Logic. VIII, 539 pages. 2004.

Vol. 3047: F. Oquendo, B. Warboys, R. Morrison (Eds.), Software Architecture. X, 279 pages. 2004.

Vol. 3046: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.J.K. Tan, O. Gervasi (Eds.), Computational Science and Its Applications – ICCSA 2004. LIII, 1016 pages. 2004.

Vol. 3045: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.J.K. Tan, O. Gervasi (Eds.), Computational Science and Its Applications – ICCSA 2004. LIII, 1040 pages. 2004.

Vol. 3044: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.J.K. Tan, O. Gervasi (Eds.), Computational Science and Its Applications – ICCSA 2004. LIII, 1140 pages. 2004.

Vol. 3043: A. Laganà, M.L. Gavrilova, V. Kumar, Y. Mun, C.J.K. Tan, O. Gervasi (Eds.), Computational Science and Its Applications – ICCSA 2004. LIII, 1180 pages. 2004.

Vol. 3042: N. Mitrou, K. Kontovasilis, G.N. Rouskas, I. Iliadis, L. Merakos (Eds.), NETWORKING 2004, Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications. XXXIII, 1519 pages. 2004.

Vol. 3040: R. Conejo, M. Urretavizcaya, J.-L. Pérez-de-la-Cruz (Eds.), Current Topics in Artificial Intelligence. XIV, 689 pages. 2004. (Subseries LNAI).

Vol. 3039: M. Bubak, G.D.v. Albada, P.M.A. Sloot, J.J. Dongarra (Eds.), Computational Science - ICCS 2004. LXVI, 1271 pages. 2004.

Vol. 3038: M. Bubak, G.D.v. Albada, P.M.A. Sloot, J.J. Dongarra (Eds.), Computational Science - ICCS 2004. LXVI, 1311 pages. 2004.

Vol. 3037: M. Bubak, G.D.v. Albada, P.M.A. Sloot, J.J. Dongarra (Eds.), Computational Science - ICCS 2004. LXVI, 745 pages. 2004.

Vol. 3036: M. Bubak, G.D.v. Albada, P.M.A. Sloot, J.J. Dongarra (Eds.), Computational Science - ICCS 2004. LXVI, 713 pages. 2004.

Vol. 3035: M.A. Wimmer (Ed.), Knowledge Management in Electronic Government. XII, 326 pages. 2004. (Subseries LNAI).

Vol. 3034: J. Favela, E. Menasalvas, E. Chávez (Eds.), Advances in Web Intelligence. XIII, 227 pages. 2004. (Subseries LNAI).

Vol. 3033: M. Li, X.-H. Sun, Q. Deng, J. Ni (Eds.), Grid and Cooperative Computing. XXXVIII, 1076 pages. 2004.

Vol. 3032: M. Li, X.-H. Sun, Q. Deng, J. Ni (Eds.), Grid and Cooperative Computing. XXXVII, 1112 pages. 2004.

Vol. 3031: A. Butz, A. Krüger, P. Olivier (Eds.), Smart Graphics. X, 165 pages. 2004.

Vol. 3030: P. Giorgini, B. Henderson-Sellers, M. Winikoff (Eds.), Agent-Oriented Information Systems. XIV, 207 pages. 2004. (Subseries LNAI).

Vol. 3029: B. Orchard, C. Yang, M. Ali (Eds.), Innovations in Applied Artificial Intelligence. XXI, 1272 pages. 2004. (Subseries LNAI).

Vol. 3028: D. Neuenschwander, Probabilistic and Statistical Methods in Cryptology. X, 158 pages. 2004.

Vol. 3027: C. Cachin, J. Camenisch (Eds.), Advances in Cryptology - EUROCRYPT 2004. XI, 628 pages. 2004.

Vol. 3026: C.V. Ramamoorthy, R. Lee, K.W. Lee (Eds.), Software Engineering Research and Applications. XV, 377 pages. 2004.

Vol. 3025: G.A. Vouros, T. Panayiotopoulos (Eds.), Methods and Applications of Artificial Intelligence. XV, 546 pages. 2004. (Subseries LNAI).

Vol. 3024: T. Pajdla, J. Matas (Eds.), Computer Vision - ECCV 2004. XXVIII, 621 pages. 2004.

Vol. 3023: T. Pajdla, J. Matas (Eds.), Computer Vision - ECCV 2004. XXVIII, 611 pages. 2004.

Vol. 3022: T. Pajdla, J. Matas (Eds.), Computer Vision - ECCV 2004. XXVIII, 621 pages. 2004.

Table of Contents

Software Variability: Requirements

Testing Reusable Software

Feature Modelling

Aspect-Oriented Software Development

Component and Service Composition

Code Level Reuse

Libraries, Classification, and Retrieval

Model-Based Approaches

Transformation and Generation

Requirements

Supporting Software Variability by Reusing Generic Incomplete Models at the Requirements Specification Stage*

Rebeca P. Díaz Redondo, Martín López Nores, José J. Pazos Arias,
Ana Fernández Vilas, Jorge García Duque, Alberto Gil Solla,
Belén Barragáns Martínez, and Manuel Ramos Cabrer

Department of Telematic Engineering. University of Vigo.
36200, Vigo. Spain. Fax number: +34 986 812116
{rebeca, mlnores, jose, avilas, jgd, agil, belen,
mramos}@det.uvigo.es

Abstract. Selecting components that satisfy a given set of requirements is a key problem in software reuse, especially in reusing between different domains of functionality. This concern has been treated in the ARIFS methodology, which provides an environment to reuse partial and formal requirements specifications, managing the variability implicit in their incompleteness. In this paper, we define *generic incomplete specifications*, to introduce an explicit source of variability that allows reusing models across different domains, accommodating them to operate in multiple contexts. An extended formal basis is defined to deal with these tasks, that entails improvements in the reuse environment.

1 Introduction

Reusability has been widely suggested to be a key to improve software development productivity and quality, especially if reuse is tackled at early stages of the life cycle. However, while many references in the literature focus on reusing at late stages (basically code), there is little evidence to suggest that reusing at early ones is widely practiced. The ARIFS methodology [1, 2] (*Approximate Retrieval of Incomplete and Formal Specifications*) deals precisely with this concern, providing a suitable framework for reusing formal requirements specifications. It combines the well-known advantages of reusing at the requirements specification stage with the benefits of a formal approach, avoiding ambiguity problems derived from natural language.

In an incremental process, the elements found at intermediate stages are characterized by their *incompleteness*. ARIFS is involved with a formal treatment of the *variability* inherent to this incompleteness. It covers the prospects of vertical reuse, i.e., reuse within the same domain of functionality. In this paper, we go one step further, to fully comply with the usual definition of software variability as "*the ability of a software artifact to be changed or customized to be used in multiple contexts*" [12]. Our proposal is to support an explicit form of variability that allows reusing models across different domains. We do this by defining *generic formal requirements specifications* and extending ARIFS' formal basis to classify and retrieve them.

* Partially supported by PGIDT01PX132203PR project (Xunta de Galicia)

J. Bosch and C. Krueger (Eds.): ICSR 2004, LNCS 3107, pp. 1–10, 2004.

The paper is organized as follows. Sections 2 and 3 describe the ARIFS reuse environment and the life cycle where it is applied. Section 4 introduces generic components and defines the basis for their classification and retrieval. Section 5 includes a simple example showing the advantages of the new proposal, compared to the original reuse environment. Section 6 comments other relevant works on the paper's scope and describes our future lines of work. A brief summary is finally given in Section 7.

2 The SCTL-MUS Methodology

SCTL-MUS [7] is a formal and incremental methodology for the development of distributed reactive systems, whose life cycle (Fig. 1) captures the usual way in which developers are given the specification of a system: starting from a rough idea of the desired functionality, this is successively refined until the specification is complete.

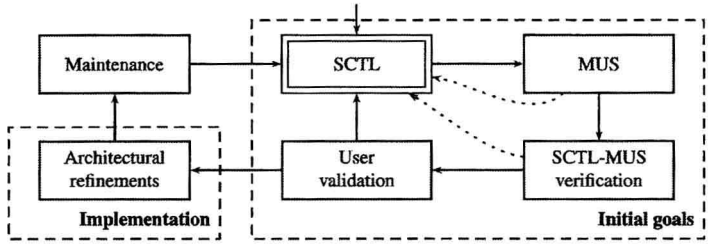


Fig. 1. The SCTL-MUS life cycle

Specifications are elaborated at the "*Initial goals*" stage. The requirements stated up to any given moment (in box "*SCTL*") are used to synthesize a model or prototype (in box "*MUS*"). When it is ready, a model checker verifies the global objectives (*properties*) of the system ("*SCTL-MUS verification*") to find if the model satisfies them; if it cannot satisfy them, neither in the current iteration nor in future ones (*inconsistency*); or if it does not satisfy them, but it may in future iterations (*incompleteness*). In case of inconsistencies, the user is given suggestions to solve them. Then, by animating the MUS prototype ("*User validation*"), the user can decide whether the current specification is already complete or more iterations are needed. Upon completion, the system enters the ("*Implementation*") stage. Here, the prototype is translated into the LOTOS process algebra [5] to obtain an initial architecture, that is progressively refined until allowing its semi-automatic translation into code language.

SCTL-MUS combines three formal description techniques. First, the many-valued logic SCTL (*Simple Causal Temporal Logic*) is used to express the functional requirements of a system, following the pattern *Premise* $\Rightarrow\otimes$ *Consequence*. Depending on the temporal operator, the consequence is assessed on the states where the premise is defined ($\Rightarrow$ or "*simultaneously*"), their predecessors ($\Rightarrow\odot$ or "*previously*") or their successors ($\Rightarrow\bigcirc$ or "*next*"). Second, the graph formalism MUS (*Model of Unspecified States*) is a variation of traditional labeled transitions systems (LTS), used to obtain system prototypes in terms of states and event-triggered transitions. Finally, the LOTOS process algebra is used to express the architecture of the developed system.

In an incremental specification process, it is essential to differentiate functional features that have been specified to be *false* (impossible) from those about which

nothing has been said yet. SCTL introduces this concept of *unspecification* by adding a third value to the logic: an event can be specified to be *true* (1) or *false* (0), being *not-yet-specified* ($\frac{1}{2}$) by default. Analogously, MUS graphs support unspecification in both states and events, thus being adequate to model incomplete specifications.

Unspecification entails that intermediate models have freedom degrees, so that they can evolve into potentially many systems. Therefore, unspecification implies variability. The incremental specification process makes the system under development lose unspecification at each iteration, by evolving *not-yet-specified* elements into *true* or *false* ones, to eventually become the desired system. ARIFS was defined to take advantage of this implicit form of variability for the purposes of reuse.

3 The ARIFS Reuse Environment

ARIFS provides for the classification, retrieval and adaptation of reusable components in SCTL-MUS. Its objectives are twofold: (i) to save specification and synthesis efforts, by reusing suitable incomplete specifications; and (ii) to reduce the extensive resources needed to check (at every single iteration) the consistency of medium to large specifications, by reusing previously obtained formal verification results.

For these purposes, a reusable component is defined by: (a) its functional specification, expressed by a set of SCTL requirements and modeled by the corresponding MUS graph; (b) verification information summarizing the levels of satisfaction of the properties that have been verified on it; and (c) an interface or profile, used for classification and retrieval, that is automatically extracted from the functional specification.

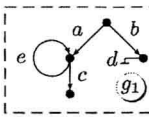
Classifying Reusable Components. The idea behind classification is that "*the closer two reusable components are classified, the more functional similarities they have*". According to this, we have defined four criteria to identify semantic and structural similarities [2]. This section describes those relevant for this paper: the TC^{∞} and NE^{∞} functions, whose results are included in the profile of the reusable components.

The TC^{∞} function offers a *semantic* viewpoint of reusable components. It associates with every MUS graph $g \in \mathbb{G}$ a set $TC^{\infty}(g)$ that contains sequences of events linked to its evolution paths. It follows the traditional complete-traces semantics [11], although it also reflects non-finite evolution paths and considers both *true* and *false* events, to differentiate these from *not-yet-specified* ones. On the other hand, NE^{∞} offers a *structural* viewpoint: given a MUS graph $g \in \mathbb{G}$, it returns a set $NE^{\infty}(g)$ reflecting the number of transitions that the model makes through every evolution path.

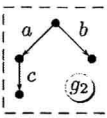
For each $\mathcal{O} \in \{TC^{\infty}, NE^{\infty}\}$, an equivalence relation $=_{\mathcal{O}} \in \mathbb{G} \times \mathbb{G}$ is defined, such that $g =_{\mathcal{O}} g' \Leftrightarrow \mathcal{O}(g) = \mathcal{O}(g')$. This organizes reusable components into equivalence classes of components indistinguishable using $\mathcal{O}$-observations. There is also a preorder relation $\sqsubseteq_{\mathcal{O}} \in \mathbb{G} \times \mathbb{G}$, given by $g \sqsubseteq_{\mathcal{O}} g' \Leftrightarrow \mathcal{O}(g) \sqsubseteq \mathcal{O}(g')$ that establishes a partial order between equivalence classes, so that $(\mathbb{G}, \sqsubseteq_{\mathcal{O}})$ is a *partially ordered set.*

Example 1. The following figure shows the result of applying the TC^{∞} and NE^{∞} functions to a MUS graph, g_1, that has four different evolution paths: (i) it can evolve through event b to a final state where d is not possible; (ii) it can reach another final state through events a and c; (iii) it can enter an endless loop through event a and an

infinite number of events e; and (iv), it can reach a final state through event a, any number of events e and then event c.



$TC^{\infty}(g_1) = (b\neg d, ac, a(e)+, a(e)+c)$
$NE^{\infty}(g_1) = (1, 2, (2)+, (3)+)$



$TC^{\infty}(g_2) = (ac, b)$
$NE^{\infty}(g_2) = (1, 2)$

All these possibilities are reflected in $TC^{\infty}(g_1)$, where the + notation means that the sequences of events inside the parenthesis can be repeated any number of times. From the NE^{∞} point of view, for the evolution paths enumerated above, the system makes one, two, at least two and at least three transitions, respectively. On another hand, it is easy to see that the MUS graph g_2 is NE^{∞}- and TC^{∞}-included in g_1.

The relationships among MUS graphs extrapolate directly to the corresponding reusable components. So, they allow organizing a repository of reusable components in two different lattices: the NE^{∞} lattice, intended for structural similarities (horizontal reuse) and the TC^{∞} lattice, for semantic ones (vertical reuse).

Retrieving Reusable Components. The variability commented at the end of Sect. 2 allows the retrieval process to be based on functional proximity instead of on functional equivalence. Taking this into account, ARIFS performs *approximate retrievals*. Queries represent functional patterns of the SCTL statements that describe the system being developed. Actually, they are defined in the same terms as the profiles of the reusable components (NE^{∞} and TC^{∞} patterns), so the equivalence and partial orderings defined above also hold between queries and reusable components.

For efficiency reasons, the retrieval process is split in two steps. In the first phase, the adjacent components of the query in the NE^{∞} and TC^{∞} lattices are located. In the second, the search is refined by quantifying the functional differences between those components and the query, in order to minimize the adaptation efforts. In the case of NE^{∞}, differences are measured in terms of a numerical distance. As for TC^{∞}, two functions assess semantic differences: the *functional consensus* and the *functional adaptation*. A detailed description of these aspects can be found at [2].

Finally, the user is given the choice between the component closest to the query according to NE^{∞} criterion, and the closest according to TC^{∞}. This selection cannot be fully automated because the adaptation cost is expressed differently in each case.

Reusing Verification Information. The idea behind the reuse of verification information in ARIFS is that useful information about the system being developed can be deduced from reusable components that are functionally close to it. With this aim, each reusable component stores verification results about every property that has been verified on it. We have proved that, for any two TC^{∞}-related components, interesting verification information can be extracted from one to the other, helping to reduce the amount of formal verification tasks throughout the specification process. The results of this work are summarized in [1].

4 Defining Generic Components

Practical experience with ARIFS has revealed some features that can be improved. As shown in Fig. 2, an NE^{∞}-based retrieval may return wildly different components, hard to adapt to the desired functionality despite being NE^{∞}-equivalent to the query.

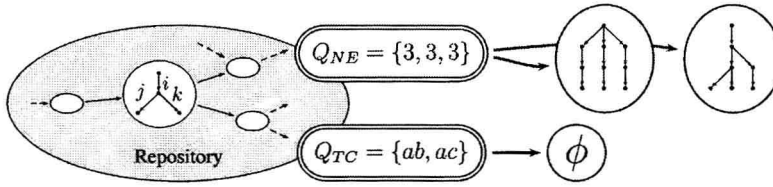


Fig. 2. Limitations of NE^{∞} and TC^{∞} criteria

On the other hand, the TC^{∞} criterion can only return components whose transitions are labeled the same way as the query. Thus, in the situation depicted in Fig. 2, the TC^{∞} search using the pattern $\{ab, ac\}$ would not find any suitable component. However, it is easy to notice that one of the components in the repository would be TC^{∞}-equivalent to the query under the mapping $(a \rightarrow i, b \rightarrow j, c \rightarrow k)$. What is more, any property R' verified on that component gives the same verification results as R on the current specification, where R' is obtained from R by the same mapping. So, the verification information linked to that component is potentially useful for the system being developed. However, it can not be reused with TC^{∞}, because this criterion is too linked to the domain of functionality of every particular component.

Generic components are introduced to address these problems. They have the same information as classic components (Section 3), but with the functionality and the verified properties expressed in terms of *meta-events*. Meta-events are identifiers such that different meta-events of a generic component must represent different events. To deal with them, we introduce a new criterion: MT^{∞}.

MT^{∞} associates with every MUS graph $g \in \mathbb{G}$ a set $MT^{\infty}(g)$ that contains sequences of meta-events linked to its evolution paths. Two graphs g and g' are MT^{∞}-equivalent ($g =^{\infty}_{\mathrm{MT}} g'$) iff a one-to-one mapping between the actions of g and g' exists such that, having done the mapping, $g^{map} =^{\infty}_{\mathrm{TC}} g'$. Analogously, a graph g is MT^{∞}-included in another graph g' ($g \sqsubseteq^{\infty}_{\mathrm{MT}} g'$) iff all the actions of g can be mapped to a different action of g' in a way that, having done the mapping, $g^{map} \sqsubseteq^{\infty}_{\mathrm{TC}} g'$ (see Fig. 3).

An ordering exists (Eq. (1)) such that, if two components are TC^{∞}-related, they are MT^{∞}- and NE^{∞}-related in the same way: $C_i \sqsubseteq^{\infty}_{\mathrm{TC}} C_j \Rightarrow C_i \sqsubseteq^{\infty}_{\mathrm{MT}} C_j \Rightarrow C_i \sqsubseteq^{\infty}_{\mathrm{NE}} C_j$.

$$NE^{\infty} \preceq MT^{\infty} \preceq TC^{\infty} \tag{1}$$

Our proposal in this paper is to organize the repository in a single lattice of generic components, using the MT^{∞} relations. MT^{∞} merges structural and semantic viewpoints in a convenient way: it is much less permissive than NE^{∞} identifying structural similarities, and abstracts the domain of functionality by considering generic actions. This introduces an explicit form of variability in the alphabet of actions of a reusable component. We also propose MT^{∞} to be the only criterion to conduct the retrieval process, automating the decision of which component to reuse; and the profile of reusable components to be formed by the results of the NE^{∞} and MT^{∞} functions.