Mario Bravetti
Manuel Núñez
Gianluigi Zavattaro (Eds.)

LNCS 4184

Web Services and Formal Methods

Third International Workshop, WS-FM 2006
Vienna, Austria, September 2006
Proceedings

Springer

Mario Bravetti Manuel Núñez
Gianluigi Zavattaro (Eds.)

Web Services and Formal Methods

Third International Workshop, WS-FM 2006
Vienna, Austria, September 8-9, 2006
Proceedings

Springer

Volume Editors

Mario Bravetti
Università di Bologna
Department of Computer Science
Via Sacchi 3, 47023 Cesena (FC), Italy
E-mail: bravetti@cs.unibo.it

Manuel Núñez
Facultad de Informática (UCM)
28040 Madrid, Spain
E-mail: mn@sip.ucm.es

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

Library of Congress Control Number: 2006931574

CR Subject Classification (1998): D.2.4, C.2.4, F.3, D.4, C.4, K.4.4, C.2

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-38862-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-38862-3 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11841197 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 4184

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

Editorial Board

Lecture Notes in Computer Science

For information about Vols. 1–4061

please contact your bookseller or Springer

Vol. 4185: R. Mizoguchi, Z. Shi, F. Giunchiglia (Eds.), The Semantic Web – ASWC 2006. XX, 778 pages. 2006.

Vol. 4184: M. Bravetti, M. Núñez, G. Zavattaro (Eds.), Web Services and Formal Methods. X, 289 pages. 2006.

Vol. 4180: M. Kohlhase, OMDoc – An Open Markup Format for Mathematical Documents [version 1.2]. XIX, 428 pages. 2006. (Sublibrary LNAI).

Vol. 4176: S.K. Katsikas, J. Lopez, M. Backes, S. Gritzalis, B. Preneel (Eds.), Information Security. XIV, 548 pages. 2006.

Vol. 4168: Y. Azar, T. Erlebach (Eds.), Algorithms – ESA 2006. XVIII, 843 pages. 2006.

Vol. 4163: H. Bersini, J. Carneiro (Eds.), Artificial Immune Systems. XII, 460 pages. 2006.

Vol. 4162: R. Královič, P. Urzyczyn (Eds.), Mathematical Foundations of Computer Science 2006. XV, 814 pages. 2006.

Vol. 4159: J. Ma, H. Jin, L.T. Yang, J.J.-P. Tsai (Eds.), Ubiquitous Intelligence and Computing. XXII, 1190 pages. 2006.

Vol. 4158: L.T. Yang, H. Jin, J. Ma, T. Ungerer (Eds.), Autonomic and Trusted Computing. XIV, 613 pages. 2006.

Vol. 4156: S. Amer-Yahia, Z. Bellahsène, E. Hunt, R. Unland, J.X. Yu (Eds.), Database and XML Technologies. IX, 123 pages. 2006.

Vol. 4155: O. Stock, M. Schaerf (Eds.), Reasoning, Action and Interaction in AI Theories and Systems. XVIII, 343 pages. 2006. (Sublibrary LNAI).

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

Vol. 4152: Y. Manolopoulos, J. Pokorný, T. Sellis (Eds.), Advances in Databases and Information Systems. XV, 448 pages. 2006.

Vol. 4151: A. Iglesias, N. Takayama (Eds.), Mathematical Software - ICMS 2006. XVII, 452 pages. 2006.

Vol. 4150: M. Dorigo, L.M. Gambardella, M. Birattari, A. Martinoli, R. Poli, T. Stützle (Eds.), Ant Colony Optimization and Swarm Intelligence. XVI, 526 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. 4136: R.A. Schmidt (Ed.), Relations and Kleene Algebra in Computer Science. XI, 433 pages. 2006.

Vol. 4135: C.S. Calude, M.J. Dinneen, G. Păun, G. Rozenberg, S. Stepney (Eds.), Unconventional Computation. X, 267 pages. 2006.

Vol. 4134: K. Yi (Ed.), Static Analysis. XIII, 443 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. 4128: W.E. Nagel, W.V. Walter, W. Lehner (Eds.), Euro-Par 2006 Parallel Processing. XXXIII, 1221 pages. 2006.

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

Vol. 4126: P. Barahona, F. Bry, E. Franconi, N. Henze, U. Sattler, Reasoning Web. X, 269 pages. 2006.

Vol. 4124: H. de Meer, J.P. G. Sterbenz (Eds.), Self-Organizing Systems. XIV, 261 pages. 2006.

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

Vol. 4119: C. Dony, J.L. Knudsen, A. Romanovsky, A. Tripathi (Eds.), Advanced Topics in Exception Handling Components. X, 302 pages. 2006.

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

Vol. 4116: R. De Prisco, M. Yung (Eds.), Security and Cryptography for Networks. XI, 366 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. 4110: J. Díaz, K. Jansen, J.D.P. Rolim, U. Zwick (Eds.), Approximation, Randomization, and Combinatorial Optimization. XII, 522 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. 4095: S. Nolfi, G. Baldassare, R. Calabretta, D. Marocco, D. Parisi, J.C. T. Hallam, O. Miglino, J.-A. Meyer (Eds.), From Animals
to Animats 9. XV, 869 pages. 2006. (Sublibrary LNAI).

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. 4087: F. Schwenker, S. Marinai (Eds.), Artificial Neural Networks in Pattern Recognition. IX, 299 pages. 2006. (Sublibrary LNAI).

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

Vol. 4084: M.A. Wimmer, H.J. Scholl, Å. Grönlund, K.V. Andersen (Eds.), Electronic Government. XV, 353 pages. 2006.

Vol. 4083: S. Fischer-Hübner, S. Furnell, C. Lambrinoudakis (Eds.), Trust and Privacy in Digital Business. XIII, 243 pages. 2006.

Vol. 4082: K. Bauknecht, B. Pröll, H. Werthner (Eds.), E-Commerce and Web Technologies. XIII, 243 pages. 2006.

Vol. 4081: A. M. Tjoa, J. Trujillo (Eds.), Data Warehousing and Knowledge Discovery. XVII, 578 pages. 2006.

Vol. 4080: S. Bressan, J. Küng, R. Wagner (Eds.), Database and Expert Systems Applications. XXI, 959 pages. 2006.

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

Vol. 4077: M.-S. Kim, K. Shimada (Eds.), Geometric Modeling and Processing - GMP 2006. 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).

Preface

This volume contains the proceedings of the international workshop WS-FM (Web Services and Formal Methods) held at Vienna University of Technology, Vienna, Austria, during September 8-9, 2006.

The International Workshop on Web Services and Formal Methods aims to bring together researchers working on Web services and formal methods in order to activate a fruitful collaboration in this direction of research. This, potentially, could also have a great impact on the current standardization phase of Web service technologies. The main topics of the conference include: protocols and standards for WS (SOAP, WSDL, UDDI, etc.); languages and descripion methodologies for Choreography/Orchestration/Workflow (BPML, XLANG and BizTalk, WSFL, WS-BPEL, etc.); coordination techniques for WS (transactions, agreement, coordination services, etc.); semantics-based dynamic WS discovery services (based on Semantic Web/ontology techniques or other semantic theories); security, performance evaluation and quality of service of WS; semi-structured data and XML related technologies; comparisons with different related technologies/approaches.

This third edition of the workshop (WS-FM 2006) featured 15 papers selected among 40 submissions after a rigorous review process by international reviewers and three invited talks by Wil van der Aalst (Eindhoven University of Technology, The Netherlands), Roberto Bruni (University of Pisa, Italy) and Schahram Dustdar (Vienna University of Technology, Austria). These contributions brought an additional dimension to the technical and the scientific merit of the workshop. This volume of the proceedings contains the 15 selected papers and three papers related to the invited talks.

WS-FM 2006 was held as an official event of "The Process Modelling Group" (a resarch group which promotes study and experimentation in business processes whose members mainly work in academia, for software companies or as part of standards bodies) and in conjunction with the 4th International Conference on Business Process Management (BPM 2006).

We owe special thanks to all members of the Program Committee of WS-FM 2006 and their sub-referees for their work. Finally, our thanks go to the University of Technology of Vienna for hosting the workshop and for their support in the workshop organization.

September 2006

Mario Bravetti
Manuel Núñez
Gianluigi Zavattaro

Organization

Program Committee

Co-chairs

Mario Bravetti	University of Bologna (Italy)
Gianluigi Zavattaro	University of Bologna (Italy)

Board of "The Process Modelling Group"

Wil van der Aalst	Eindhoven Univ. of Technology, The Netherlands
Rob van Glabbeek	NICTA, Sydney, Australia
Keith Harrison-Broninski	Role Modellers Ltd.
Robin Milner	Cambridge University, UK
Roger Whitehead	Office Futures

Other PC Members

Marco Aiello	University of Trento (Italy)
Farhad Arbab	CWI, The Netherlands
Matteo Baldoni	University of Turin, Italy
Jean-Pierre Banatre	University of Rennes 1 and INRIA, France
Boualem Benatallah	University of New South Wales, Australia
Karthik Bhargavan	Microsoft Research Cambridge, UK
Roberto Bruni	University of Pisa, Italy
Michael Butler	University of Southampton, UK
Fabio Casati	HP Labs, USA
Rocco De Nicola	University of Florence, Italy
Marlon Dumas	Queensland University of Technology, Australia
Schahram Dustdar	Vienna University of Technology, Austria
Gianluigi Ferrari	University of Pisa, Italy
Jose Luiz Fiadeiro	University of Leicester, UK
Stefania Gnesi	CNR Pisa, Italy
Reiko Heckel	University of Leicester, UK
Kohei Honda	Queen Mary, University of London, UK
Nickolas Kavantzas	Oracle Co., USA
Leila Kloul	Université de Versailles, France
Cosimo Laneve	University of Bologna, Italy
Mark Little	Arjuna Technologies Limited, UK
Natalia López	University Complutense of Madrid, Spain
Roberto Lucchi	University of Bologna, Italy
Jeff Magee	Imperial College London, UK

Fabio Martinelli	CNR Pisa, Italy
Manuel Mazzara	University of Bolzano, Italy
Ugo Montanari	University of Pisa, Italy
Shin Nakajima	National Institute of Informatics and JST, Japan
Manuel Nunez	University Complutense of Madrid, Spain
Fernando Pelayo	University of Castilla-La Mancha, Albacete, Spain
Marco Pistore	University of Trento, Italy
Wolfgang Reisig	Humboldt University, Berlin, Germany
Vladimiro Sassone	University of Sussex, UK
Marjan Sirjani	Tehran University, Iran
Friedrich Vogt	Technical University of Hamburg-Harburg, Germany
Martin Wirsing	Ludwig Maximilians University Munich, Germany

Additional Referees

Massimo Bartoletti	Stephanie Kemper	Marinella Petrocchi
Marzia Buscemi	Natallia Kokash	Stephan Reiff-Marganiec
Samuele Carpineti	Alessandro Lapadula	Bilel Remmache
Diego Cazorla	Luis Llana	Shamim Ripon
Corina Cirstea	Mieke Massink	Ismael Rodriguez
Sara Corfini	Ilaria Matteucci	Francesco Tiezzi
Fernando Cuartero	Hernan Melgratti	Angelo Troina
Alberto de la Encina	Sebastian Menge	Emilio Tuosto
Berndt Farwer	Mercedes G. Merayo	Divakar Yadav
Fatemeh Ghassemi	Leonardo Mezzina	Uwe Zdun
Dieter Gollmann	Luca Padovani	

Organizing Committee

Chair

Manuel Núñez	University Complutense of Madrid, Spain

Local Chair

Friedrich Neubarth	Austrian Research Institute for Artificial Intelligence

Other Members

Mario Bravetti	University of Bologna, Italy
Gregorio Díaz	Universidad Castilla-La Mancha, Spain
Alberto de la Encina	Universidad Complutense de Madrid, Spain
Roberto Lucchi	University of Bologna, Italy
Mercedes G. Merayo	Universidad Complutense de Madrid, Spain
Gianluigi Zavattaro	University of Bologna, Italy

Table of Contents

I Invited Papers

II Contributed Papers

DecSerFlow: Towards a Truly Declarative Service Flow Language

W.M.P. van der Aalst and M. Pesic

Department of Information Systems, Eindhoven University of Technology,
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl, m.pesic@tm.tue.nl

Abstract. The need for process support in the context of web services has triggered the development of many languages, systems, and standards. Industry has been developing software solutions and proposing standards such as BPEL, while researchers have been advocating the use of formal methods such as Petri nets and π-calculus. The languages developed for *service flows*, i.e., process specification languages for web services, have adopted many concepts from classical workflow management systems. As a result, these languages are rather procedural and this does not fit well with the autonomous nature of services. Therefore, we propose *DecSerFlow* as a *Declarative Service Flow Language*. DecSerFlow can be used to specify, enact, and monitor service flows. The language is extendible (i.e., constructs can be added without changing the engine or semantical basis) and can be used to enforce or to check the conformance of service flows. Although the language has an appealing graphical representation, it is grounded in temporal logic.

Keywords: Service flows, web services, workflow management, flexibility, temporal logic.

1 Introduction

The *Business Process Execution Language for Web Services* (BPEL4WS, or BPEL for short) has become the de-facto standard for implementing processes based on web services [7]. Systems such as Oracle BPEL Process Manager, IBM WebSphere Application Server Enterprise, IBM WebSphere Studio Application Developer Integration Edition, and Microsoft BizTalk Server 2004 support BPEL, thus illustrating the practical relevance of this language. Although intended as a language for connecting web services, its application is not limited to cross-organizational processes. It is expected that in the near future a wide variety of process-aware information systems [8] will be realized using BPEL. Whilst being a powerful language, BPEL is of a *procedural nature* and not very different from classical workflow languages e.g., the languages used by systems such as Staffware, COSA, SAP Workflow, and IBM WebSphere MQ Workflow (formerly know as FlowMark). Also other languages proposed in the context of web services are of a procedural nature, e.g., the *Web Services Choreography*

M. Bravetti, M. Nuñes, and G. Zavattaro (Eds.): WS-FM 2006, LNCS 4184, pp. 1–23, 2006.

Description Language (WS-CDL) [16]. In this paper, we will not discuss these languages in detail. The interested reader is referred to [2,3,20] for a critical review of languages like BPEL. Instead, we will demonstrate that it is possible to use a more declarative style of specification by introducing *DecSerFlow*: a *Declarative Service Flow Language*.

To explain the difference between a procedural style and a declarative style of modeling, we use a simple example. Suppose that there are two activities A and B. Both *can* be executed multiple times but they *exclude* each other, i.e., after the first occurrence of A it is not allowed to do B anymore and after the first occurrence of B it is not allowed to do A. The following execution sequences are possible based on this verbal description: *[ ]* (the empty execution sequence), *[A]*, *[B]*, *[A,A]*, *[B,B]*, etc. In a *procedural* language it is difficult to specify the above process without implicitly introducing additional assumptions and constraints. In a procedural language one typically needs to make a choice with respect to whether no activities are to be executed, only A activities are to be executed, or only B activities are to be executed. Moreover, the number of times A or B needs to be executed also has to be decided. This means that one or more decision activities need to be executed before the execution of "real" activities can start. (Note that this is related to the Deferred Choice pattern described in [4].) The introduction of these decision activities typically leads to an over-specification of the process. Designers may be tempted to make this decision before the actual execution of the first A or B. This triggers the following two questions: (1) "How is this decision made?" and (2) "When is this decision made?". The designer may even remove the choice altogether and simply state that one can only do A activities. Using a more *declarative* style can avoid this over-specification. For example, in *Linear Temporal Logic* (LTL) [11,12,13] one can write $\neg(\Diamond A \wedge \Diamond B)$. This means that it cannot be the case that eventually A is executed and that eventually B is executed. This shows that a very compact LTL expression ($\neg(\Diamond A \wedge \Diamond B)$) can describe exactly what is needed without forcing the designer to specify more than strictly needed. Unfortunately, languages like LTL are difficult to use for non-experts. Therefore, we have developed a graphical language (DecSerFlow) that allows for the easy specification of processes in a declarative manner. DecSerFlow is mapped onto LTL. The innovative aspects of our approach based on DecSerFlow are:

- DecSerFlow allows for a *declarative style* of modeling which is highly relevant in the context of service flows (unlike languages like BPEL).
- Through the *graphical representation* of DecSerFlow this language is easy to use and we avoid the problems of textual languages like LTL.
- We use LTL not only for the verification of model properties: we also use the LTL formulas generated by DecSerFlow to *dynamically monitor services* and to realize an *enactment engine*.
- DecSerFlow is an extendible language (i.e., we supply an editor to extend the language with user-defined graphical constructs without the need to modify any part of the system).

- DecSerFlow can be used to specify two types of constraints: *hard* constraints and *soft* constraints. Hard constraints are enforced by the engine while soft constraints are only used to warn before the violation takes place and to monitor observed violations.

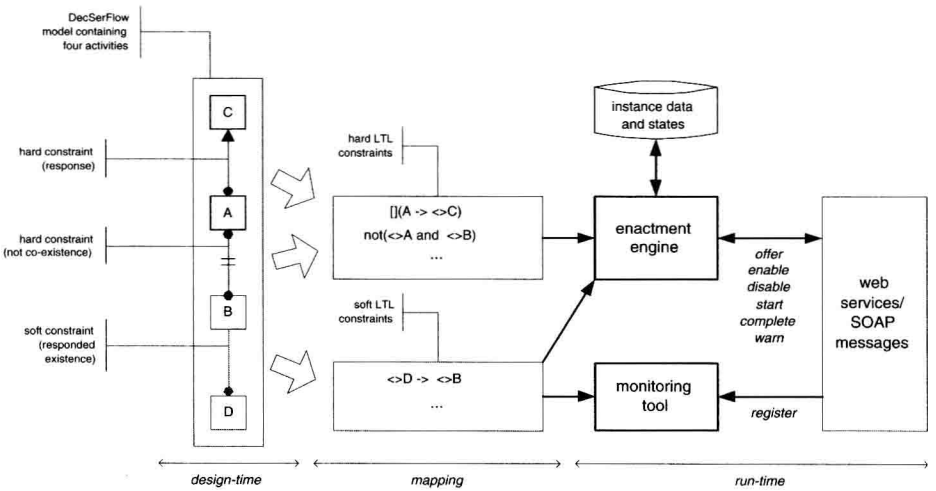

Fig. 1. Overview of the role played by DecSerFlow in supporting services flows

Figure 1 provides an overview of the way we envision DecSerFlow to be used. At design-time, a graphical model is made using the DecSerFlow notation. (Note that at design-time users can also add new modeling elements - types of constraints.) The left-hand side of Figure 1 shows a process composed of four activities, A, B, C, and D. Moreover, three constraints are shown. The connection between A and C means that any occurrence of A should eventually be followed by at least one occurrence of C (i.e., $\Box(A \rightarrow \Diamond C)$ in LTL terms). The connection between A and B means that it cannot be the case that eventually A is executed and that eventually B is executed. This is the constraint described before, i.e., $\neg(\Diamond A \wedge \Diamond B)$ in LTL terms. The last constraint connecting D and B is a soft constraint. This constraint states that any occurrence of D implies also the occurrence of B (before or after the occurrence of D), e.g., *[B,D,D,D,D]*, *[D,D,D,B]*, and *[B,B,B]* are valid executions. The LTL formulation of this constraint is $\Diamond D \rightarrow \Diamond B$.

As Figure 1 shows, it is possible to automatically map the graphical model onto LTL formulas. These formulas can be used by the enactment engine to control the service flow, e.g., on the basis of hard constraints the engine can allow or prohibit certain activities and on the basis of soft constraints warnings can be issued. The soft constraints can also be used by the monitoring tool to detect and analyze violations.

Currently, we have implemented a graphical editor and the mapping of the editor to LTL. This editor supports user-defined notations as described before. We are currently investigating different ways to enact LTL formulas and in this paper we described our current efforts. Although we do not elaborate this this in this paper, our implementation will also incorporate data as is show in Figure 1. Data is used for routing purposes by making constraints data dependent, i.e., a constraint only applies if its guard evaluates to true. Moreover, in the context of the ProM (Process Mining) framework [6,18] we have developed an LTL checker [1] to compare actual behavior with specified behavior. The actual behavior can be recorded by a dedicated process engine. However, it can also be obtained by monitoring SOAP messages as described in [3].

The approach described in Figure 1 is not limited to service flows. It can be applied in any context where *autonomous* entities are executing activities. These autonomous entities can be other organizations but also people or groups of people. This is the reason that DecSerFlow has a "sister language" named *ConDec* which aims at supporting teamwork and workflow flexibility [17]. Both languages/applications share the same concepts and tools.

The remainder of this paper is organized as follows. Section 2 introduces the DecSerFlow language. Then, a non-trivial example is given in Section 3. Section 4 discusses different ways to construct an enactment (and monitoring) engine based on DecSerFlow. Finally, Section 5 concludes the paper by discussing different research directions.

2 DecSerFlow: A Declarative Service Flow Language

Languages such as *Linear Temporal Logic* (LTL) [11,12,13] allow for the a more declarative style of modeling. These languages include temporal operators such as next-time ($\bigcirc F$), eventually ($\Diamond F$), always ($\Box F$), and until ($F \sqcup G$). However, such languages are difficult to read. Therefore, we define an extendible graphical syntax for some typical constraints encountered in service flows. The combination of this graphical language and the mapping of this graphical language to LTL forms the *Declarative Service Flow (DecSerFlow) Language* . We propose DecSerFlow for the *specification of a single service, simple service compositions, and more complex choreographies.*

Developing a model in DecSerFlow starts with creating activities. The notion of an activity is like in any other workflow-like language, i.e., an activity is atomic and corresponds to a logical unit of work. However, the nature of the *relations between activities* in DecSerFlow can be quite different than in traditional procedural workflow languages (like Petri nets and BPEL). For example, places between activities in a Petri net describe causal dependencies and can be used to specify sequential, parallel, alternative, and iterative routing. Using such mechanisms it is both possible and necessary to strictly define *how* the flow will be executed. We refer to relations between activities in DecSerFlow as *constraints.* Each of the constraints represents a policy (or a business rule). At any point in time during the execution of a service, each constraint evaluates to

true or *false*. This value can change during the execution. If a constraint has the value *true*, the referring policy is fulfilled. If a constraint has the value *false*, the policy is violated. The execution of a service is *correct* (according to the DecSerFlow model) at some point in time if all constraints (from the DecSerFlow model) evaluate to *true*. Similarly, a service has *completed correctly* if at the end of the execution all constraints evaluate to *true*. The goal of the execution of any DecSerFlow model is not to keep the values of all constraints *true* at all times during the execution. A constraint which has the value *false* during the execution is not considered an error. Consider for example the LTL expression $\Box(A \longrightarrow \Diamond B)$ where A and B are activities, i.e., each execution of A is eventually followed by B. Initially (before any activity is executed), this LTL expression evaluates to *true*. After executing A the LTL expression evaluates to *false* and this value remains *false* until B is executed. This illustrates that a constraint may be temporarily violated. However, the goal is to end the service execution in a state where all constraints evaluate to *true*.

To create constraints in DecSerFlow we use *constraint templates*. Each constraint template consists of a formula written in LTL and a graphical representation of the formula. An example is the "response constraint", which is denoted by a special arc connecting two activities A and B. The semantics of such an arc connecting A and B are given by the LTL expression $\Box(A \longrightarrow \Diamond B)$, i.e., any execution of A is eventually followed by (at least one) execution of B. We have developed a starting set of constraint templates and we will use these templates to create a DecSerFlow model. This set of templates is inspired by a collection of specification patterns for model checking and other finite-state verification tools [9]. Constraint templates define various types of dependencies between activities at an abstract level. Once defined, a template can be reused to specify constraints between activities in various DecSerFlow models. It is fairly easy to change, remove and add templates, which makes DecSerFlow an "open language" that can evolve and be extended according to the demands from different domains.[1] In the initial set of constraint templates we distinguish three groups: (1) "existence", (2) "relation", and (3) "negation" templates. Because a template assigns a graphical representation to an LTL formula, we will refer to such a template as a formula.

Before giving an overview of the initial set of formulas and their notation, we give a small example explaining the basic idea. Figure 2 shows a DecSerFlow model consisting of four activities: A, B, C, and D. Each activity is tagged with a constraint describing the number of times the activity should be executed, these are the so-called "existence formulas". The arc between A and B is an example of a "relation formula" and corresponds to the LTL expression discussed before: $\Box(\ A \longrightarrow \Diamond\ B\ )$. The connection between C and D denotes another "relation formula": $\Diamond\ D \longrightarrow \Diamond\ C$, i.e., if D is executed at least once, C is also executed at least once. The connection between B and C denotes a "negation formula"

[1] Note that we have developed a graphical editor for DecSerFlow that supports the creation of user defined templates, i.e., the user can define the graphical representation of a generic constraint and give its corresponding semantics in terms of LTL.

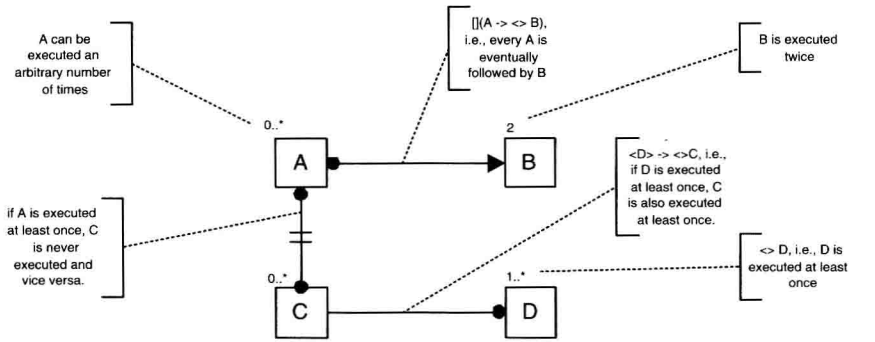

Fig. 2. A DecSerFlow model showing some example notations

(the LTL expression is not show here). Note that it is not easy to provide a classical procedural model (e.g., a Petri net) that allows for all behaviour modeled Figure 2.

Existence Formulas. Figure 3 shows the so-called "existence formulas". These formulas define the possible number of executions (cardinality) of an activity. For example, the first formula is called *existence*. The name and the formula heading are shown in the first column. From this, we can see that it takes one parameter (A), which is the name of an activity. The body of the formula is written in LTL and can be seen in the second column. In this case the LTL expression $\Diamond(activity == A)$ ensures that the activity given as the parameter A will execute at least once. Note that we write $\Diamond(activity == A)$ rather than $\Diamond(A)$. The reason is that in a state we also want to access other properties, i.e., not just the activity name but also information on data, time, and resources. Therefore, we need to use a slightly more verbose notation ($activity == A$). The diagram in the third column is the graphical representation of the formula, which is assigned to the template. Parameter A is an activity and it is represented as a square with the name of the activity. The constraint is represented by a cardinality annotation above the square. In this case the cardinality is at least one, which is represented by *1..**. The first group of existence formulas are of the cardinality "N or more", denoted by *N..**. Next, the formula *absence* ensures that the activity should never execute in the service. The group of formulas with names *absence_N* uses negations of *existence_N* to specify that an activity can be executed at most *N-1* times. The last group of existence formulas defines an exact number of executions of an activity. For example, if a constraint is defined based on the formula *exactly_2*, the referring activity has to be executed exactly two times in the service.

Relation Formulas. Figure 4 shows the so-called "relations formulas". While an "existence formula" describes the cardinality of one activity, a "relation formula" defines relation(s) (dependencies) between two activities. All relation formulas