Studies in Computational Intelligence 150

Roger Lee (Ed.)

Software Engineering Research, Management and Applications

Springer

Studies in Computational Intelligence, Volume 150

Editor-in-Chief
Prof. Janusz Kacprzyk
Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warsaw
Poland
E-mail: kacprzyk@ibspan.waw.pl

Further volumes of this series can be found on our homepage: springer.com

Vol. 128. Fatos Xhafa and Ajith Abraham (Eds.)
Metaheuristics for Scheduling in Industrial and Manufacturing Applications, 2008
ISBN 978-3-540-78984-0

Vol. 129. Natalio Krasnogor, Giuseppe Nicosia, Mario Pavone and David Pelta (Eds.)
Nature Inspired Cooperative Strategies for Optimization (NICSO 2007), 2008
ISBN 978-3-540-78986-4

Vol. 130. Richi Nayak, Nikhil Ichalkaranje and Lakhmi C. Jain (Eds.)
Evolution of the Web in Artificial Intelligence Environments, 2008
ISBN 978-3-540-79139-3

Vol. 131. Roger Lee and Haeng-Kon Kim (Eds.)
Computer and Information Science, 2008
ISBN 978-3-540-79186-7

Vol. 132. Danil Prokhorov (Ed.)
Computational Intelligence in Automotive Applications, 2008
ISBN 978-3-540-79256-7

Vol. 133. Manuel Graña and Richard J. Duro (Eds.)
Computational Intelligence for Remote Sensing, 2008
ISBN 978-3-540-79352-6

Vol. 134. Ngoc Thanh Nguyen and Radoslaw Katarzyniak (Eds.)
New Challenges in Applied Intelligence Technologies, 2008
ISBN 978-3-540-79354-0

Vol. 135. Hsinchun Chen and Christopher C. Yang (Eds.)
Intelligence and Security Informatics, 2008
ISBN 978-3-540-69207-2

Vol. 136. Carlos Cotta, Marc Sevaux and Kenneth Sörensen (Eds.)
Adaptive and Multilevel Metaheuristics, 2008
ISBN 978-3-540-79437-0

Vol. 137. Lakhmi C. Jain, Mika Sato-Ilic, Maria Virvou, George A. Tsihrintzis, Valentina Emilia Balas and Canicious Abeynayake (Eds.)
Computational Intelligence Paradigms, 2008
ISBN 978-3-540-79473-8

Vol. 138. Bruno Apolloni, Witold Pedrycz, Simone Bassis and Dario Malchiodi
The Puzzle of Granular Computing, 2008
ISBN 978-3-540-79863-7

Vol. 139. Jan Drugowitsch
Design and Analysis of Learning Classifier Systems, 2008
ISBN 978-3-540-79865-1

Vol. 140. Nadia Magnenat-Thalmann, Lakhmi C. Jain and N. Ichalkaranje (Eds.)
New Advances in Virtual Humans, 2008
ISBN 978-3-540-79867-5

Vol. 141. Christa Sommerer, Lakhmi C. Jain and Laurent Mignonneau (Eds.)
The Art and Science of Interface and Interaction Design (Vol. 1), 2008
ISBN 978-3-540-79869-9

Vol. 142. George A. Tsihrintzis, Maria Virvou, Robert J. Howlett and Lakhmi C. Jain (Eds.)
New Directions in Intelligent Interactive Multimedia, 2008
ISBN 978-3-540-68126-7

Vol. 143. Uday K. Chakraborty (Ed.)
Advances in Differential Evolution, 2008
ISBN 978-3-540-68827-3

Vol. 144. Andreas Fink and Franz Rothlauf (Eds.)
Advances in Computational Intelligence in Transport, Logistics, and Supply Chain Management, 2008
ISBN 978-3-540-69024-5

Vol. 145. Mikhail Ju. Moshkov, Marcin Piliszczuk and Beata Zielosko
Partial Covers, Reducts and Decision Rules in Rough Sets, 2008
ISBN 978-3-540-69027-6

Vol. 146. Fatos Xhafa and Ajith Abraham (Eds.)
Metaheuristics for Scheduling in Distributed Computing Environments, 2008
ISBN 978-3-540-69260-7

Vol. 147. Oliver Kramer
Self-Adaptive Heuristics for Evolutionary Computation, 2008
ISBN 978-3-540-69280-5

Vol. 148. Philipp Limbourg
Dependability Modelling under Uncertainty, 2008
ISBN 978-3-540-69286-7

Vol. 149. Roger Lee (Ed.)
Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2008
ISBN 978-3-540-70559-8

Vol. 150. Roger Lee (Ed.)
Software Engineering Research, Management and Applications, 2008
ISBN 978-3-540-70774-5

Roger Lee
(Ed.)

Software Engineering Research, Management and Applications

Springer

Prof. Roger Lee
Computer Science Department
Central Michigan University
Pearce Hall 413
Mt. Pleasant, MI 48859
USA
Email: lee1ry@cmich.edu

ISBN 978-3-540-70774-5 e-ISBN 978-3-540-70561-1

DOI 10.1007/978-3-540-70561-1

Studies in Computational Intelligence ISSN 1860949X

Library of Congress Control Number: 2008930466

Typeset & Cover Design: Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed in acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Preface

The 6th ACIS International Conference on Software Engineering, Research, Management and Applications (SERA 2008) was held in Prague in the Czech Republic on August 20–22. SERA '08 featured excellent theoretical and practical contributions in the areas of formal methods and tools, requirements engineering, software process models, communication systems and networks, software quality and evaluation, software engineering, networks and mobile computing, parallel/distributed computing, software testing, reuse and metrics, database retrieval, computer security, software architectures and modeling. Our conference officers selected the best 17 papers from those papers accepted for presentation at the conference in order to publish them in this volume. The papers were chosen based on review scores submitted by members or the program committee, and underwent further rounds of rigorous review.

In chapter 1, Luciana Akemi Burgareli et al. discuss the challenges of managing variability in software development product lines. The authors present a strategy for variability management, and perform a case test of that strategy on an existing product line of Brazilian Satellite Launcher Vehicle software.

In chapter 2, Ivan Garcia et al. look into the use of questionnaire-based appraisal. They then apply this method to business models, and recommend for use with small and medium sized enterprises.

In chapter 3, Abbas Heydarnoori investigates the challenges of the software deployment process. He proposes that certain automatic tools and techniques be used to streamline the deployment process, and he then provides a general overview of these tools and techniques.

In chapter 4, James H. Hill and Aniruddha Gokhale present a simple technique, based on baseline profiling, for searching or deployments of both distributed real-time and embedded systems that improve performance along a simple quality-of-service concern. They perform experiments to verify the validity of their technique, and present the observations of those experiments to recommend new deployments.

In chapter 5, Lucia Kapová et al. address the problems associated with the common methods of transforming Business Process Modeling Notation (BPMN) into Business Process Executable Language (BPEL), as related to Service-oriented Architecture. The authors propose an enhancement of the existing transformation algorithm which provides a complete transformation, while preserving a larger part of the intent of the original BPMN descriptions that previously possible.

In chapter 6, Gang Huang et al. investigate the challenges in Software Architecture (SA) associated with the flattening of hierarchical SA models in order to transform platform-independent models into platform-specific models. The authors note the problem of lost comprehensibility, redundancy and consistency in the transformation process, and recommend a new systematic approach that preserves these qualities.

In chapter 7, Otso Kassinen et al. present a case study that provides practical guidelines for Simbian OS software development based on a three-year mobile software research project on the criterion of networking middleware and collaborative applications. The authors present observations that advocate the use of platform-independent solutions, the minimization of project dependencies and the representation of complex activities in human-readable form.

In chapter 8, Muhammad Bilal Latif and Aamer Nadeem investigate the extraction of a Finite State Machine (FSM) as associated with the writing of requirements specifications for safety-critical systems. They discuss the challenge of automatic FSM generation from Z-specifications as caused by difficulties in identifying and extracting pre- and postconditions. The authors then present an automated approach to the separation of this data, as a solution to this problem, and provide tools and experimentation to support their suggestion.

In chapter 9, Tomás Martínez-Ruiz et al. suggest a SPEM extension that will support the variability implied in a software process line. They provide for new methods in their extension, in order to allow for the variability needed in a software process line.

In chapter 10, Noor Hasnah Moin and Huda Zuhrah Ab. Halim propose a hybrid Genetic Algorithm to determine the order of requests to be scheduled in the data routing of a telecommunications network. The authors then discuss the design of three algorithms based on the Variable Neighbor Search. They conclude by comparing the performance of these algorithms on a set of data from the OR library.

In chapter 11, Iulian Ober and Younes Lakhrissi propose the use of events as a first class concept for the compositions of software components. They show how the approach can be applied to any language based on concurrent opponents and illustrate their claim with examples.

In chapter 12, Annie Ressouche et al. investigate the challenges of creating automatic specification and verification tools for synchronous languages. They design a new specification model based on a reactive synchronous approach. In practice they design and implement a special purpose language that allows for dealing both with large systems and formal validation.

In chapter 13, Haldor Samset and Rolv Bræk readdress the notion of active services in the context of Service-oriented Architecture. In their paper, the authors explain how active services and their behaviors can be described for publication and discovery.

In chapter 14, Ilie Şavga and Michael Rudolf show how the use of a history of structural component changes enables automatic adaptation of existing adaptation specifications.

In chapter 15, Dimitrios Settas and Ioannis Stamelos propose the Dependency Structure Matrix (DSM) as a method that visualizes and analyzes the dependencies between related attributes of software project management antipatterns in order to reduce complexity and interdependence. The authors exemplify their solution with a

DSM of 25 attributes and 16 related software project management antipatterns that appear in the literature on the Web.

In chapter 16, Gang Shen proposes a practical curriculum for an embedded systems software engineering undergraduate program. The curriculum advocates a proactive learning setting in close cooperation with the relevant industry.

In chapter 17, Yoshiyuki Shinkawa proposes a formal model verification process for UML use case models. The author then describes the results of a test example performed on a supermarket checkout system.

It is our sincere hope that this volume provides stimulation and inspiration, and that it be used as a foundation for works yet to come.

May 2008 Roger Lee

Contents

List of Contributors

Rolv Bræk
Norwegian University of Science, Norway
rolv@item.ntnu.no

Tomáš Bureš
Charles University, Czech Republic
bures@dsrg.mff.cuni.cz

Luciana Akemi Burgareli
Aeronautics and Space Institute, Brazil
luciana@iae.cta.br

Mauricio G.V. Ferreira
Institute for Space Research, Brazil
mauricio@ccs.inpe.br

Daniel Gaffé
University of Nice Sophia Antipolis CNRS, France
daniel.gaffe@unice.fr

Ivan Garcia
Technological University of the Mixtec Region, Mexico
ivan@mixteco.utm.mx

Félix García
University of Castilla-La Mancha, Spain
felix.garcia@uclm.es

Aniruddha Gokhale
Vanderbilt University, USA
a.gokhale@vanderbilt.edu

Huda Zuhrah Ab. Halim
University of Malaya, Malaysia
noor_hasnah@um.edu.my

Erkki Harjula
University of Oulu, Finland
erkki.harjula@ee.oulu.fi

Abbas Heydarnoori
University of Waterloo, Canada
aheydarnoori@uwaterloo.ca

James H. Hill
Vanderbilt University, USA
j.hill@vanderbilt.edu

Petr Hnětynka
Charles University, Czech Republic
hnetynka@dsrg.mff.cuni.cz

Gang Huang
Peking University, China
huanggang@sei.pku.edu.cn

Lucia Kapová
Charles University, Czech Republic
kapova@dsrg.mff.cuni.cz

Otso Kassinen
University of Oulu, Finland
otso.kassinen@ee.oulu.fi

Timo Koskela
University of Oulu, Finland
timo.koskela@ee.oulu.fi

Younes Lakhrissi
Université de Toulouse - IRIT, France
iulian.ober@irit.fr

Muhammad Bilal Latif
Mohammad Ali Jinnah University, Pakistan
m.bilal.latif@gmail.com

Tomás Martínez-Ruiz
University of Castilla-La Mancha, Soain
tomas.martinez@uclm.es

Hong Mei
Peking University, China
meih@sei.pku.edu.cn

Selma S.S. Melnikoff
Polytechnic School of University of Sao Paulo, Brazil
selma.melnikoff@poli.usp.br

Noor Hasnah Moin
University of Malaya, Malaysia
noor_hasnah@um.edu.my

Aamer Nadeem
Mohammad Ali Jinnah University, Pakistan
aamern@acm.org

Iulian Ober
Université de Toulouse - IRIT, France
iulian.ober@irit.fr

Carla Pacheco
Technological University of the Mixtec Region, Mexico
leninca@mixteco.utm.mx

Mario Piattini
University of Castilla-La Mancha, Spain
mario.piattni@uclm.es

Annie Ṙessouche
INRIA Sophia Antipolis - Méditerranée, France
Annie.Ressouche@sophia.inria.fr

Valérie Roy
CMA Ecole des Mines Sophia Antipolis, France
vr@cma.ensmp.fr

Michael Rudolf
Technische Universität Dresden, Germany
s0600108@inf.tu-dresden.de

Haldor Samset
Norwegian University of Science, Norway
haldors@item.ntnu.no

Ilie Şavga
Technische Universität Dresden, Germany
is13@inf.tu-dresden.de

Dimitrios Settas
Aristotle University of Thessaloniki, Greece
dsettas@csd.auth.gr

Gang Shen
Huazhong University of Science and Technology, China
gang_shen@yahoo.com

Yoshiyuki Shinkawa
Ryukoku University, Japan
shinkawa@rins.ryukoku.ac.jp

Ioannis Stamelos
Aristotle University of Thessaloniki, Greece
dsettas@csd.auth.gr

Pavel Sumano
Technological University of the Mixtec Region, Mexico
sumanoop@mixtli.utm.mx

Yanchun Sun
Peking University, China
sunyc@sei.pku.edu.cn

Jie Yang
Peking University, China
yangj@sei.pku.edu.cn

Mika Ylianttila
University of Oulu, Finland
mika.ylianttila@ee.oulu.fi

13. Dick, J., Faivre, A.: Automating the generation and sequencing of test cases from model-based specifications. In: Larsen, P.G., Woodcock, J.C.P. (eds.) FME 1993. LNCS, vol. 670, pp. 268–284. Springer, Berlin (1993)
14. Sun, J., Dong, J.S.: Extracting FSMs from Object-Z specifications with history invariants. In: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems ICECCS 2005, pp. 96–105. IEEE Computer Society, Washington (2005)
15. Murray, L., Carrington, D., MacColl, I., McDonald, J., Strooper, P.: Formal derivation of finite state machines for class testing. In: P. Bowen, J., Fett, A., Hinchey, M.G. (eds.) ZUM 1998. LNCS, vol. 1493, pp. 42–59. Springer, Berlin (1998)
16. Stocks, P., Carrington, D.: A framework for specification-based testing. IEEE Transactions on Software Engineering 22(11), 777–793 (1996)
17. Hierons, R.M.: Testing from a Z specification. Software Testing, Verification and Reliability 7(1), 19–33 (1997)
18. Ashraf, A., Nadeem, A.: Automating the generation of test cases from Object-Z specifications. In: Proceedings of the 30th Annual International Computer Software and Applications Software COMPSAC 2006, pp. 101–104. IEEE Computer Society, Washington (2006)
19. Murray, L., Carrington, D., MacColl, I., Strooper, P.: TinMan - A Test Derivation and Management Tool for Specification-Based Class Testing. In: Proceedings of the 32nd International Conference on Technology of Object-Oriented Languages, pp. 222–233. IEEE Computer Society, Washington (1999)
20. Huaikou, M., Ling, L.: A test class framework for generating test cases from Z specifications. In: Proceedings of the 6th IEEE International Conference on Complex Computer Systems ICECCS 2000, pp. 164–171. IEEE Computer Society, Washington (2000)
21. Ling, L., Huaikou, M., Xuede, Z.: A framework for specification-based class testing. In: Proceedings of the 8th IEEE International Conference on Complex Computer Systems, ICECCS 2002, pp. 153–162. IEEE Computer Society, Washington (2002)
22. Sun, J., Dong, J.S.: Design synthesis from interaction and state-based specifications. IEEE Transactions on Software Engineering 32(2), 349–364 (2006)
23. Jacky, J.: The way of Z: Practical Programming with Formal Methods. Cambridge University Press, New York (1996)

Towards a SPEM v2.0 Extension to Define Process Lines Variability Mechanisms

Tomás Martínez-Ruiz[1], Félix García[1], and Mario Piattini[1]

Alarcos Research Group, Information and Technology Systems Department
Escuela Superior de Informática, University of Castilla-La Mancha
Paseo de la Universidad, 4, 13071 Ciudad Real, Spain
{tomas.martinez,felix.garcia,mario.piattini}@uclm.es

Summary. Software organizations need to adapt their processes to each new project. Although Software Process Lines are the most suitable approach for the design of processes which are adapted to different contexts, SPEM does not include the appropriate mechanisms for modelling them. The objective of this paper is to suggest a SPEM extension which will support the variability implied in a Software Process Line. New variability mechanisms based on the use of Variation Points and Variants, by means of which the variability necessary in a Process Line is represented, have been proposed. The new mechanisms that we shall introduce into SPEM, will allow it to model Software Process Lines. From these lines, the generation of processes adapted to each context will simplify the selection of the appropriate variants for each variation point.

Keywords: Software Process Lines, SPEM, Variability.

1 Introduction

Software development organizations are currently interested in increasing their competitiveness and quality levels. In order to achieve this target, they need to have well-defined processes. For this reason, various process evaluation and improvement models have been proposed, such as CMMI, ISO/IEC 15504, SCAMPI, MoProSoft, EvalProSoft or the COMPETISOFT Methodological Framework processes.

However, the diversity of enterprises, projects and contexts in which processes take place is too diverse. For example, in Spain, the Civil Service Ministry requires the that totality of the software that it uses be developed using Métrica v3 [7], which is not used in the projects of other official bodies. This makes the statement *Just as there are no two identical software projects, there are no two identical software processes in the world* [5] clear. In this respect, it is difficult to apply defined generic process models in organizations without having previously adapted them to the specific situations in which they are going to be developed [18].

In the adaptation process of a software process, a set which is very similar to the original processes is developed, although these processes are not very different from each other. All these processes, which are virtually identical, make up a

R. Lee (Ed.): Soft. Eng. Research, Management & Applications, SCI 150, pp. 115–130, 2008.
springerlink.com

Software Process Family. In a process family, processes can make good use of their similarities and exploit their differences, in a manner similar to that of Software Product Lines [13].

In order to generate the differences that distinguish each process of a software process line, it is necessary to facilitate mechanisms through which this variation in the processes can be defined. However, SPEM (Software process Engineering Metamodel) [12], which has been confirmed as being the appropriate Metamodel with which to represent software processes does not, at present have the appropriate variability mechanisms to allow the generation of processes following the Software Process Line approach to take place.

In this paper, new variability mechanisms in SPEM are proposed, which contribute with the functionality necessary to model Software Process Lines. Besides this introduction, in Sect. 2 a state of the art in software product lines variability mechanisms is presented. Section 3 contains a summary of SPEM and an analysis of its current variability mechanisms, so that we can go on to show our proposal for variability mechanisms in Sect. 4. Finally, in Sect. 5 our conclusions and future work are presented.

2 State of the Art: Variability in Product Lines

To the best of our knowledge no other works exist in which the product line approach has been applied to form modelling variability in software processes, which constitutes the topic of this work.

However, some approaches dealing with variability in software product lines exist, which it is useful to analyze within the context of the work presented here.

Within the relevant literature, the majority of the variability mechanisms proposed for Software Product Lines are based on variation points and variants.

A variation point is the place in which variability occurs [2], whereas variants are the concrete elements that are placed at the variation points, each of which implements this variability in a different way.

In [17], after an analysis of the various approaches through which the variability in Software Product lines can be modelled, it was determined that the use of variants and variation points presents, amongst other advantages, the possibility of adding new variability implementations, by adding new variants. Furthermore, according to [1], it is possible to specify dependences and constraints between these elements.

In [15] a solution which is also based on variations, variation points and their relationships is proposed. This solution furthermore defines dependences between variation points and includes three abstraction levels in which to model variability. These components are included in the COVAMOF framework, which is validated by means of an experiment.

The PULSE methodology defines the five basic elements of a software product line: variation points, variants, their relations, constraints and a mechanism with which to generate products. It also includes a Decision Model in a high level of abstraction to define system variants [14].

Van der Hoeck proposes an approach based on XML schemas which model th system structural breakdown in a incremental way. One notable characteristic c this mechanism is that it allows us to manage variability owing to the evolutio of versions of the product itself [4].

In [3], an idea based on a set of patterns with which to build, manage an manipulate variation points is presented.

XVLC [19] is a methodology that adds extensions to UML in order to mod variation points and variants, and it furthermore specifies that the choice of variant affects software architecture and distinguishes those variants that affec the whole system in a crosscutting manner.

Software Process Lines variability mechanisms can be designed by using th ideas from these proposals as a starting point. These mechanisms can be adde to the process definition of SPEM to permit the Process Lines specification.

3 Software Process Engineering MetaModel

SPEM (Software Process Engineering Metamodel) [12] is the Object Manage ment Group (OMG) proposal through which to represent software processe and methods. It is based on other OMG models, such as, MOF (Meta Objec Facility) [10] and UML (Unified Modelling Language).

Recently the new 2.0 version of this Standard has been developed, whic contributes significant new capabilities to the latest version. The Metamodel i divided into seven packages, with a hierarchical structure Fig. 1.

The following is a summary of the contents of each package illustrated in Fig. 1

- **Core:** this contains all those classes and abstractions which are part of th metamodel base.
- **Process Structure:** this sub-package contains the elements which are nec essary to define process models.
- **Process Behaviour:** this models process behaviour.
- **Managed Content:** this contains the elements which are needed to manag textual method descriptions.
- **Method Content:** this sub-package includes the concepts with which t define methods.
- **Process with Methods:** this sub-package permits method and process in tegration.
- **Method Plugin:** this sub-package contains the metamodel concepts wit which to design and manage reusable and configurable methods and proces libraries and repositories. It also includes certain variability elements an allows us to define the granularly extended process.

SPEM includes graphic notation, but uses UML association relationship SPEM's notation allows us to represent and visualize diagrams with processe and methods.

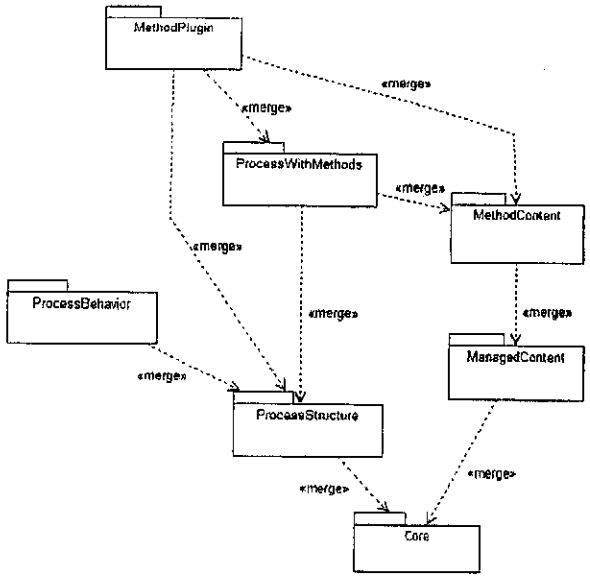

Fig. 1. Structure of the SPEM 2.0 Metamodel [12]

3.1 Variability in SPEM

SPEM v2.0 defines variability within its *MethodPlugin* package. It includes the classes which are necessary to allow variability in methods and processes, by means of defining the abstract *VariabilityElement* class and the *VariabilityType* enumeration.

The abstract *VariabilityElement* class (Fig. 2) allows us to add the variability concept to the process and the method definition of both the *ProcessStructure* and *MethodContend* packages. Given that the *Activity* (*Process Structure* package), *MethodContentElement* (*MethodContent* package) and *Section* (*ManagedContent* package) classes inherit from the VariabilityElement class, then these and their descendants acquire the "vary" capability.

By means of variation, the structure of the method can be customized without having to modify it directly.

The *VariailityType* enumeration defines the type of variability between both instances of the *VariabilityElement* class. It includes the *contributes*, *replaces*, *extends*, *extends-replaces* and *na* values (by default). The first four are now described:

Contributes is a variability relationship that allows the addition of a *VariabilityElement* to another base, without altering its original contents. This

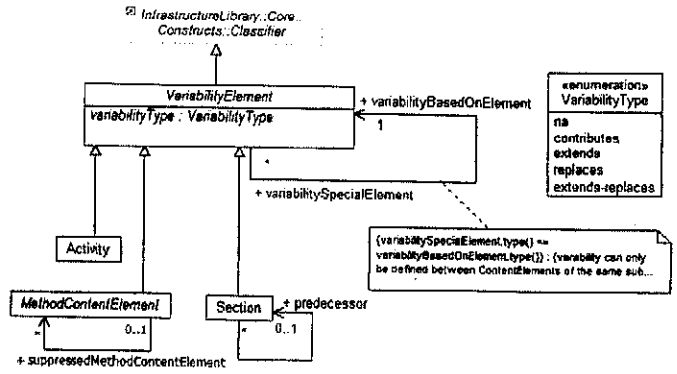

Fig. 2. *VariabilityElement* class and its relationships [12]

relationship has transitive properties. A base element must have more than one contributor.

Replaces is a variability mechanism which permits the *VariabilityElement* to be replaced by another one, without modifying its properties. A base element can only define a replaces relationship. Like contribution, the *replaces* relationship is transitive.

The **extends** relationship is an inheritance mechanism between the *VariabilityElement*. This relationship is similar to the UML *extends* relationship. This relationship is also transitive and both the contributes and replaces relationships take priority over extends.

The **extends-replaces** relationship combines the effects of both previous relationships. So while the *replace* relationship replaces all the properties of the base element, this one only replaces those values which have been redefined in the substitute element.

3.2 Limitations of the SPEM v2.0's Process Lines Variability

As a result of the analysis carried out in SPEM, certain limitations have been detected when it is used to model Software Process Lines.

1. The proposed variability in SPEM is orientated more towards changing methods than processes. In fact, the diagram in Fig. 2 shows that the *methodContent* class inherits from the *VariabilityElement* class. The methodContent class is a super-class of all the classes which permit the design of methods, and only the *Activity* class is used to model processes. Due to the fact that the processes are the entities that are executed, it will be desirable to carry out the necessary modifications/adaptations in order to execute it in the context of a project within the process, and not in the

method upon which it is based. Furthermore, we must take account that when the method content is used, the user can define the correspondence between the process and method elements, and may ignore the variability introduced into the method, so that it cannot be used.

2. This variability mechanisms do not permit the delimitation of the *core* part of the Software Process Line. As we have discussed previously, a Software Process Line is composed of a set of processes which have certain common characteristics, and which differ in other characteristics, and this makes them suitable for one context or another. However, the variability mechanisms of SPEM do not allow us to establish that a part of the process line will be common to all processes, or to limit these variations.
3. The variability mechanisms of SPEM do not guarantee that the variations introduced to generate a new process will not alter its objectives. The goals must be common to all the adjustments that are made in the process using variability mechanisms. However, because the variability mechanisms of SPEM do not allow us to define the core of a Software Process Line, it is not possible to include the objectives of the process in it and ensure that they are not modified. To avoid this, the actions of the variability mechanisms should be more limited. That is, in a process line it should be possible to specify which parts may vary and in which range of values.
4. It may be difficult to reuse SPEM v2.0 plugins with the variability mechanisms defined. In the SPEM *MethodPlugin* the *MethodPlugin* elements are defined as follows: these are extensible packages that represent a physical container for process and method packages [12]. Likewise, the *replaces* operation is used in *MethodPlugins* to replace *ContentElements* such as *RolcUse*, *TaskUse*, or *Activity* with a new variant or to change the relationships between them [12].
   This adaptation approach using plugins has the disadvantage that it is necessary to define a new plugin in order to extend that of the base which contains the process specification and adapt it to the context in which it will be enacted.
5. SPEM v2.0 does not include a specific notation with which to represent the process variability. Variability is shown by using the UML association and inheritance relationships, characterized by means of stereotypes. However, it is difficult to read some of these relations.

To sum up, due to the limitations previously listed, we believe that SPEM v2.0 variability mechanisms can be enhanced for modelling variability in Software Process Lines.

4 Specific Process Lines Variability Mechanisms

We introduce an outline of our variability mechanisms proposal with the aim of allowing SPEM v2.0 to support Software Process Lines. Likewise, we analyze their integration into the SPEM metamodel currently defined, and present an example of their use.

4.1 General Description

Our proposal is based on the mechanisms with which to manage the variability defined in Software Product Lines. Our aim is to use said mechanisms to develop others which will allow us to model variability in Software Process Lines. We list the elements that we believe to be necessary below.

1. **Variation points and variants:** in the same way as was identified in the Software Product Lines, the Variation points are the point in the process model in which the variation occurs (adapted from [14]). They will be "empty" model elements that will be "filled" with variants to allow the generation of a new process from the process line which will be adapted to the context in which they are to be developed. There will be variants of each element of the model (*VActivity*, *VWorkProductDefinition*, etc.), which can be inserted at certain variation points, and which will be dependent on the context in which the process is going to be implanted. Each variation point will have a set of associated variants and will be occupied by one or more of them.
   Due to the fact that several element types take part in a process (*RolcUse*, *WorkProductUse*, *Activity*, *TaskUse*), this distinction must also be made between the different types of variant and variation point elements.
2. **Relationships and dependencies:** Both, variation points and variants define the relationships between them. Variability is an entity which is as complex as a process and cannot be seen as a precise occurrence, but as one which will affect several part of the process structure. Dependencies will therefore allow the variations introduced into a process to be consistent. That is to say, they will ensure that if a task is carried out by a role, then both are connected.
3. **Constraints:** In addition to the relationships between the elements, we also need to take into account that constraints may appear between elements. For example, a constraint might be that a mandatory variation point must be always occupied by a variant.

4.2 New Variability Mechanisms Added to SPEM v2.0

In a way similar to that in which UML profiles extend their metamodel [6], the SPEM Metamodel can be extended to support the variability mechanism defined in the previous section. Different extensions can thus be added to the *Method Plugin* standard's package. To ensure the consistence and correct use of the new elements, OCL constraints are used [11].

First, a *LProcElement* should be designed. This is an abstraction of all the elements related to the Software Process Lines variability (Fig. 3). *LProcElement* is an abstract specification of *Classifier*, from UML packages.

The two specifications of *LProcElement* shown in Fig. 3, *VarPoint* and *Variant*, are the variation point and variant representation, respectively. Variants know the variation point for which they have been designed and which they will be able to occupy.

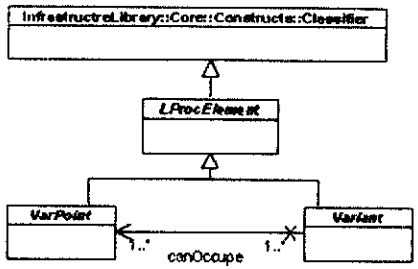

Fig. 3. Variability by means of *LProcElement*

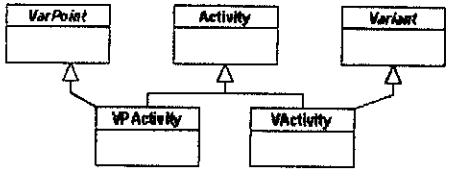

Fig. 4. *VPActivity*'s and *VActivity*'s concrete types

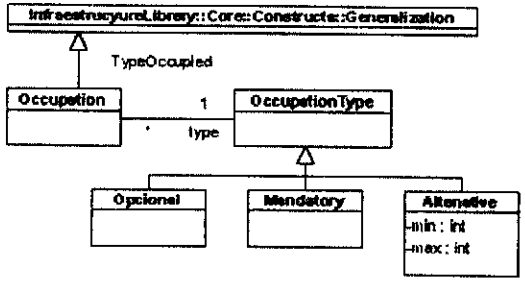

Fig. 5. *Occupation* relationship between *Variants* and *VarPoints*

The two aforementioned classes are also abstract, due to the fact that various variants and variation points of a variety of natures may exist (*Activity*, *RoleUse*). Fig. 4 shows how the variation points and variants of a concrete element were modelled. In this case, the figure shows the activity variant and variation point. As we can see, both *VPActivity* and *VActivity* are also a specification of the Activity class.

Table 1. Constraint of *Occupation* relationship between *Variation* and *Variation Point*

```
constraint Occupation inv:
    self.general.OclIsKindQf (VarPoint) and
    self.specific.OclIsKindOf (Variant)
```

Fig. 6. *Occupation* relationship arrow

Process models will contain different specifications of *VarPoint*, which will be "occupied" by the appropriate specifications of *Variant*. The relationship that binds a variation point to its variant, *Occupation*, is a UML Generalization specification (Fig. 5).

The *Occupation* relationship may in turn be of several types, which are specified by the *OccupationType*. These may be *Optional*, *Mandatory* or *Alternative* type. Given that this relationship is also a generalization, we need to define the OCL constraint to force it to be used exclusively between variants and variation points (Table 1).

Constraint of Table 2 specifies that the nature of the elements between which the *Occupation* relationship is established must be compatible and the variant is designed to occupy this variation point.

The *Occupation* relationship is drawn by using an arrow similar to that of the UML generalization, but with a circle on the back tip and a filled triangle on the front tip (Fig. 6), to differentiate it from the other SPEM relationships.

Finally, dependences can be established between elements, called *Variability-Dependences* (Fig. 7).

These are a UML dependencies specification, but can only exist between elements belonging to the Process Lines (Table 3).

Several types of variability dependencies may exist, such as *VariantToVariant*, *VarPointToVarPoint* or *VariantToVarPoint*, depending upon the element be-

Table 2. *Occupation* relationship between compatibility types constraint

```
constraint Occupation inv:
    (self.general.OclIsTypeOf (VPRoleDefinition) implies
    self.specific.OclIsTypeOf (VRoleDefinition)
    or self.general.OclIsTypeOf (VPActivity) implies
    self.specific.OclIsTypeOf(VActivity)
    or self.general.OclIsTypeOf (VPTaskDefinition) implies
    self.specific.OclIsTypeOf(VTaskDefinition)
    or self.general.OclIsTypeOf (VPWorkProduct) implies
    self.specific.OclIsTypeOf(VWorkProdut))
    and self.specific.canOccupe-¿includes(self.general)
```

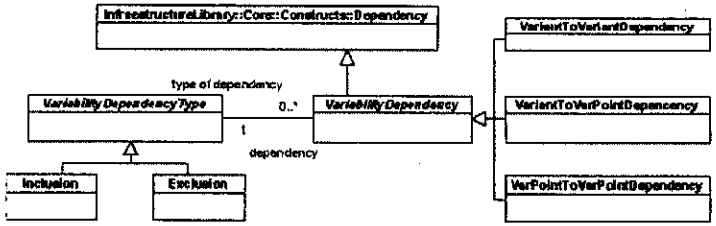

Fig. 7. *Variability Dependency* and its relationships

Table 3. *VariabilityDependency* is only available for *LProcElements*

```
constraint VariabilityDependency inv:
    self.supplier.OclIsKindOf (LProcElement) and
    self.client.OclIsKindOf (LProcElement)
```

Table 4. New *VarPoint* and *Variant* graphic icons

	Activity	WorkProductUse	RoleUse	TaskUse
Base Element	Activity	WorkProductUse	RoleUse	TaskUse
VarPoint	VPActivity	VPWorkProductUse	VPRoleUse	VPTaskUse
Variant	VActivity	VWorkProductUse	VRoleUse	VTaskUse

tween which they are defined. Each dependency is associated with its *VariabilityDependencyType*, which identifies whether it is inclusive or exclusive. For the use of the model, we also propose new icons by means of which the new variability functionality can be graphically defined. These icons are based on those used to represent those elements to which a variation capacity is given. (Table 4).

4.3 Application Example

To illustrate our proposal, in this section we show a case study carried out on the COMPETISOFT [8, 9] process model. COMPETISOFT is an iberoamerican project in which a Software Process Improvement Framework has been

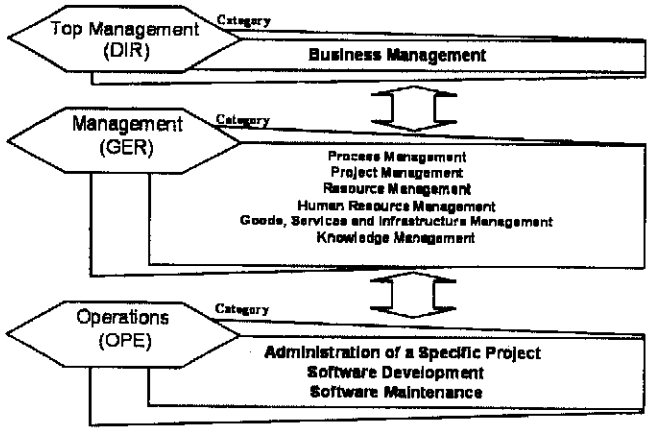

Fig. 8. Processes and categories in the COMPETISOFT process model

defined for and adapted to very small enterprises. This framework includes several process improvement proposals and is intended to be used, with the appropriate adaptations, in various iberoamerican countries. The COMPETISOFT process model is made up of ten processes which are grouped into three categories (Fig. 8). The first is Top Management and this provides the alignments which allow the organization to work. The second category, Management, includes the process, project and resource management. The Operations category deals with the practices of development projects and software maintenance.

The process model pattern includes a section called "adjustment guide" which determines possible modifications to the process which should not affect its objectives. This guide includes an introductory rough draft to the variability in the process. We shall now show how the variations in the software development process of COMPETISOFT allow said process to be modelled by using the SPEM extension described in this paper. A Process Line is thus generated, whose processes are adapted to the situations described in the adjustment guide. Without these variability mechanisms, a specific process model for each situation would be necessary.

1. Although the activities in COMPETISOFT corresponding with *Ananlysis* and *Design* have been considered seperately, in certain organisations it may be seen as expedient to fuse them, and to call the resulting activity *Anaysis and Design*. The process will thus have an activity type variation point, and a variant will take place which will contain (Fig. 9). An example which is similar to the aforementioned case occurs with the activities related to *Integration* and *Testing*.

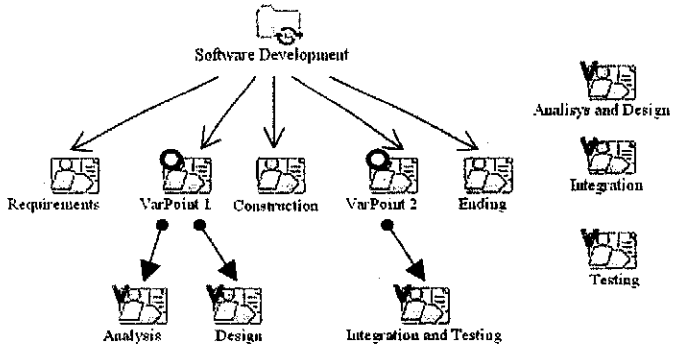

Fig. 9. Variability in the development process

Fig. 10. Variability in optional Colleague Review

2. The *Colleague Review* (sub)activity can be added to the *Construction* activity in order to verify the code of the components. This activity will be contained within a variant, which could occupy the VarPoint found in *Construction* (Fig. 10).
3. The *Requirements Specification* work product contains various products. It may, moreover, include a *User Interface Prototype* with more or less functionality, depending upon how complicated and how important the interface is. The interface should have a variable functionality which can be modelled as a variation point and can be occupied by various WorkProduct type variants (Fig. 11).
4. The *System Testing Plan* can be validated with the Client and with the *Security System*. Such (sub)activity will thus have a role type of variation point which can be occupied by the variants which correspond with the two roles previously commented upon (Fig. 12).
5. An *IEEE Standard* can be taken into account in the *Unit Testing Definition* activity. This document will be an entry variant to the task commented upon (Fig. 13).
6. A (sub)activity of the *Modification of User Interface Prototype* can be added to the *Construction* activity, in which the User, an Expert or individuals must participate. The *Construction* activity will thus have a variation point into which the variant corresponding with the aforementioned (sub)activity can be inserted. This variant will also have a variation point which can be occupied by any of the previously described roles (Fig. 14).

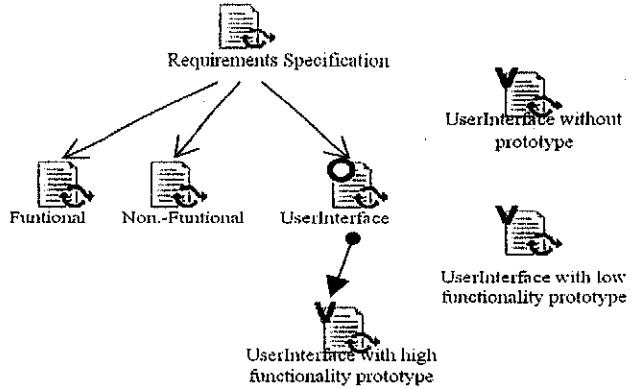

Fig. 11. Variability in the Requeriments Specification

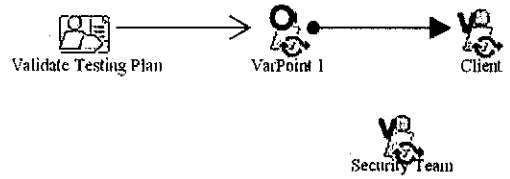

Fig. 12. Variability in the Testing Plan validation

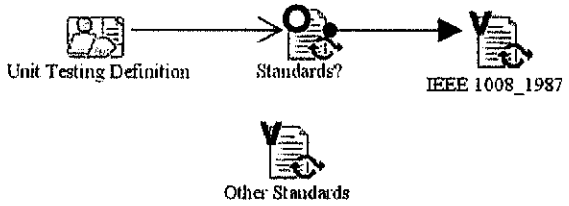

Fig. 13. Variability in the use of Standards

As this example shows, the use of the newly defined variability mechanism will allow the COMPETISOFT software development process to be adapted to various contexts. Several almost identical processes which form a part of the Software Process Lines have been generated through this adaptation.

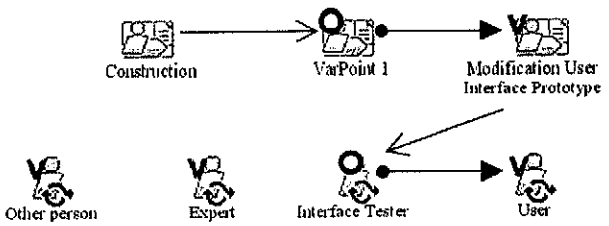

Fig. 14. Double variability with the *Modification of User Interface Prototype*

5 Conclusions and Future Work

As this work has shown, Software Process Lines approach is a powerful tool which permits the adaptation of processes to the specific conditions of an organisation and to each of the projects that it is carrying out at any given time. SPEM furthermore supposes a powerful initiative through which to model the processes which an organisation is carrying out, with the end of knowing them better, purifying them and improving them. The inclusion in SPEM of mechanisms which are appropriate to the generation of process families facilitates the creation of process models which are adapted to the particular needs of whatever is being carried out. These models are instantiated by means of the proposed mechanisms.

The approach presented in this paper allows us to manage variability from the perspective of Software Process Lines. The advantages that it offers are summarized as follows:

- This proposal is intended to measure the introduction of variability in SPEM processes, thus allowing it to define Software Process Lines.
- As with the Software Product Lines upon which they are based, these new variation mechanisms permit specification within the SPEM metamodel whose concrete parts may vary.
- By limiting the variability of the variation points, we assure that the rest of the process cannot be altered.
- Because the variants are extremely small elements, they can easily be reused at other variation points.
- This approximation includes a graphic notation with which variability can be more easily modelled in SPEM 2.0 diagrams.
- The proposed extension provides COMPETISOFT process model with a suitable notation in which the variants defined in accordance with the adjustment guide can be represented in an explicit manner.

As future work, we shall first define variability mechanisms for other constructors of the SPEM metamodel as, owing to the scope of the present work, only the core elements were considered (activities, work products, roles and tasks). Furthermore, additional empirical validation will be carried out by applying the

approach to new case studies. Other future work is to automate these variabi ity mechanisms in the EPF Composer, which supports the definition of SPEI models.

Finally, we shall also consider the aspect orientation approach to model var ability. Processes are made up of determined 'aspects', such as security, usabilit etc., which have a transversal effect upon all their tasks. Bearing in mind th similarity between this idea and that of the *'crosscutting concerns'*, defined in th Aspect Oriented programme [16], and the manner of interrelation which give place to the final code, the definition of the variability mechanisms in the Sof ware Process Lines can also be carried out by using the features of the Aspe Oriented programme.

Acknowledgement. This work is partially supported by the investigation abou Software Process Lines sponsored by Sistemas Técnicos de Loterías del Estad S.A. in the framework of the aggreement about the Innovación del Entorr Metodológico de Desarrollo y Mantenimiento de Software, and by the projec ESFINGE TIN2006-15175-C05-05 financed by spanish Science and Technolog Ministry and INGENIO financed by the Junta de Comunidades de Castilla-L Mancha, Consejería de Educación y Ciencia, PAC08-0154-9262.

References

1. Asikainen, T., Männistö, T., Soimimen, T.: Kumbang: A domain ontology f modeling variability in software product families. Advanced Engineering Informa ics 21(1), 23–40 (2007)
2. Clauß, M.: Generic modeling using uml extensions for variability. In: Workshop Domain Specific Visual Languages, Tampa Bay, Florida (2001)
3. Goedicke, M., Koellmann, C., Zdun, U.: Designing runtime variation points product line architectures: three cases. Science of Computer Programming 53(3 353–380 (2004)
4. van der Hoeck, A.: Design-time product line architectures for any-time variabilit Science of Computer Programming 53(1), 285–304 (2004)
5. Humphrey, W.S.: Managing the Software Process. Addison-Wesley, Reading (198
6. Korherr, B., List, B.: A UML 2 profile for variability models and their dependen to business processes. In: DEXA, Regensburg, Germany, pp. 829–834 (2007)
7. MAP. Métrica. versión 3. metodología de planificación, desarrollo y mantenimien de sistemas de información. Ministerio de Administraciones Públicas (2005)
8. Oktaba, H., Garcia, F., Piattini, M., Pino, F., Alquicira, C., Ruiz, F.: Softwa process improvement in small latin-american organizations: Competisoft proje IEEE Computer 40(10), 21–28 (2007)
9. Oktaba, H., Piattini, M., Pino, F., Garcia, F., Alquicira, C., Ruiz, F., Martínez, T Competisoft: A improvement strategy for small latin-american software organiz tions. In: Oktaba, H., Piattini, M. (eds.) Software Process Improvement for Sm and Medium Enterprises: Techniques and Case Studies. Idea Group Inc. (2008)
10. OMG. Meta object facility (mof) specification version 2.0. Technical report, Objc Management Group (2004)
11. OMG. The object constraint language specification- version 2.0. Technical repo Object Management Group (April 2004)

12. OMG. Software process engineering metamodel specification. Technical Report ptc/07-03-03, Object Management Group (October 2007)
13. Rombach, D.: Integrated software process and product lines. In: Li, M., Boehm, B., Osterweil, L.J. (eds.) SPW 2005. LNCS, vol. 3840, pp. 83–90. Springer, Heidelberg (2006)
14. Schmid, K., John, I.: A customizable approach to full lifecycle variability management. Science of Computer Programming 53(3), 259–284 (2004)
15. Sinnema, M., Deelstra, S.: Industrial validation of covamof. Journal of Systems and Software 49(1), 717–739 (2007)
16. Sutton, S.M.: Aspect-oriented software development and software process. In: Li, M., Boehm, B., Osterweil, L.J. (eds.) SPW 2005. LNCS, vol. 3840, pp. 177–191. Springer, Heidelberg (2006)
17. Webber, D., Gomaa, H.: Modeling variability in software product lines with the variation point model. Science of Computer Programming 53(3), 305–331 (2004)
18. Yoon, I.-C., Min, S.-Y., Bae, D.-H.: Tailoring and verifying software process. In: APSEC, Macao, pp. 202–209 (2001)
19. Zhang, H., Jarzabek, S.: XVCL: a mechanism for handling variants in software product lines. Science of Computer Programming 53(3), 381–407 (2004)

Genetic Algorithm and Varial Search for Point to Multipoin

Noor Hasnah Moin and Huda Zuhrah Ab. Halim

Institute of Mathematical Sciences
University of Malaya
50603 Kuala Lumpur
Malaysia
noor_hasnah@um.edu.my

Summary. Routing of data in a telecommunicatior vast amount of data flow in the network. Message important aspect of data routing. It is the process of each set has a single source and multiple destinations. Genetic Algorithm to determine the order of reque algorithm embeds a different Steiner Tree algorithm second part of the study involves designing three Neighborhood Search. These algorithms differ in the in searching the neighborhood. The performance of a modified set of data taken from the OR library.

1 Introduction

Communication network management is becomi the increasing network size, rapidly changing to ticular, the importance of data routing in telec longer be ignored due to the greater accessibilit munications. The need for a more reliable and importance. Message Scheduling Problem (MSP set of requests where each request has a single so [3]. Specifically, different requests may have diff tination. MSP is modeled as a Point to Multipoi PMRP solves the problem by finding the optimal [3], [11] and [6] use a combination of Genetic Al

The series *Studies in Computational Intelligence* (SCI) publishes new developments and advances in the various areas of computational intelligence – quickly and with a high quality. The intent is to cover the theory, applications, and design methods of computational intelligence, as embedded in the fields of engineering, computer science, physics and life science, as well as the methodologies behind them. The series contains monographs, lecture notes and edited volumes in computational intelligence spanning the areas of neural networks, connectionist systems, genetic algorithms, evolutionary computation, artificial intelligence, cellular automata, self-organizing systems, soft computing, fuzzy systems and hybrid intelligent systems. Critical to both contributors and readers are the short publication time and world-wide distribution – this permits a rapid and broad dissemination of research results.

The 6th ACIS International Conference on Software Engineering, Research, Management and Applications (SERA 2008) was held in Prague in the Czech Republic on August 20 – 22. SERA '08 featured excellent theoretical and practical contributions in the areas of formal methods and tools, requirements engineering, software process models, communication systems and networks, software quality and evaluation, software engineering, networks and mobile computing, parallel/distributed computing, software testing, reuse and metrics, database retrieval, computer security, software architectures and modeling. Our conference officers selected the best 17 papers from those papers accepted for presentation at the conference in order to publish them in this volume. The papers were chosen based on review scores submitted by members or the program committee, and underwent further rounds of rigorous review.

ISSN 1860-949X

ISBN 978-3-540-70774-5

Available online

springer.com