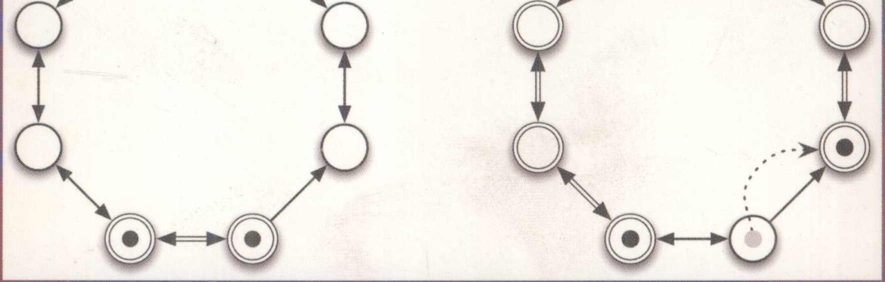
State-of-the-Art Survey

LNCS 4157

Michael Butler
Cliff Jones
Alexander Romanovsky
Elena Troubitsyna (Eds.)

Rigorous Development of Complex Fault-Tolerant Systems



Springer

Michael Butler Cliff Jones
Alexander Romanovsky Elena Troubitsyna (Eds.)

Rigorous Development of Complex Fault-Tolerant Systems

Springer

Volume Editors

Michael Butler
University of Southampton
School of Electronics and Computer Science
Highfield, Southampton SO17 1BJ, UK
E-mail: mjb@ecs.soton.ac.uk

Cliff Jones
Alexander Romanovsky
Newcastle University
School of Computing Science
Newcastle upon Tyne, NE1 7RU, UK
E-mail: {cliff.jones,alexander.romanovsky}@ncl.ac.uk

Elena Troubitsyna
Åbo Akademi University
Department of Computer Science
Lemminkäisenkatu 14 A, 20520 Turku, Finland
E-mail: etroubit@abo.fi

Library of Congress Control Number: 2006936100

CR Subject Classification (1998): C.2.4, D.1.3, D.2, D.4.5, F.2.1-2, D.3, F.3

LNCS Sublibrary: SL 2 – Programming and Software Engineering

ISSN 0302-9743
ISBN-10 3-540-48265-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-48265-9 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springer.com

Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11916246 06/3142 5 4 3 2 1 0

Lecture Notes in Computer Science 4157

Foreword

Software is the fuel of the information society. Many of our systems and applications are today controlled and/or developed in software. It is also a well known fact that many software systems have reached a level of complication, mainly because of their size, heterogeneity and distribution (and hopefully not through bad programming), that results in faults appearing which cannot be traced back easily to the code. Some of these "faults" could also be unexpected program behaviour that appears as a result of interactions between the different parts of the program; this is commonly known as complexity. The problem is that sometimes is not easy to say whether a fault is traceable to the code or whether it is due to emergent unexpected behaviour from the complex software system. Testing the code for possible faults is also very costly.

New methods, approaches, tools and techniques are needed to cope with the increasing complexity in software systems; amongst them, fault tolerance techniques and formal methods, supported by the corresponding tools, are promising solutions. This is precisely the subject of this book, which is very much welcome.

The pervasiveness of software in today's information society makes it of paramount importance, and the main objective of the Software Technologies unit of the European Commission is to support the European software and services industry so that quality software and services are developed to compete in global markets. To help in reaching this objective, it is obvious that we need to maintain and contribute to the excellence in research from universities and research organizations in this specific area.

The volume has been prepared by the partners involved in the FP6 IST-511599 RODIN project (partly funded by the European Commission), "Rigorous Open Development Environment for Complex Systems". The book brings together papers focusing on the application of rigorous design techniques to the development of fault-tolerant, software-based systems.

In RODIN complexity is mastered by design techniques (specifically formal methods) that support clear thinking and rigorous validation and verification. Coping with complexity also requires architectures that are tolerant of faults and unpredictable changes in the environment; this side is addressed by fault tolerant design techniques. The sources of complexity under study in RODIN are those caused by the environment in which the software is to operate and from the poorly conceived architectural structure.

Who should read this book? Basically, the formal methods and fault tolerance communities. The formal methods people will learn more about (and probably be fired up by) the challenging issues in design for fault tolerance, while researchers on fault tolerance will better understand how formal methods can improve way in which their techniques are developed and applied.

The European Commission, through its successive framework programs, has supported work on methods and techniques to master system complexity and achieve dependable and trustworthy systems. Recently, specifically under the 6th Framework Programme, it has called, amongst other topics, for "Principles, methodologies and tools for design, management and simulation of complex software systems" and

"Foundational and applied research to enable the creation of software systems with properties such as self-adaptability, flexibility, robustness, dependability and evolvability".

It is clear that these issues are, by no means, fully resolved. Software systems are increasingly complex, and we will need increased efforts in research just to keep up with the pace of development (based on the reflection by the Red Queen in Lewis Carroll's *Through the Looking Glass*, "in this place it takes all the running you can do, to keep in the same place"). It is time, now, for renewed efforts; this book is a pointer in that direction.

August 2006 José-Luis Fernández-Villacañas Martín

Disclaimer: The views expressed in this foreword are those of the author only and should not be construed to reflect or represent the position of the European Commission.

Preface

There was, for several decades, a split between researchers who aimed to create perfect programs by using formal methods and those who pioneered techniques for fault tolerance. Of course, the approaches actually complement each other. Fault tolerance generally copes with failures of physical components (and might in specific cases be able to guard against some sorts of design mistakes). Formal reasoning is not just about proving (under assumptions) that a given program will function perfectly; the most productive use of formalism is early on in the design process to help clean up the architecture of a system. As systems have become larger and more intimately linked both to the physical world and to human users, the design task has become far more complex. One of the goals of design must always be to reduce unnecessary complexity in resulting systems.

The editors of this book are proud to be involved in an EU (FP-6) project which specifically brings together researchers from the fault tolerance and formal methods communities. We are aware that through abstraction, refinement and proof, formal methods provide design techniques that support clear thinking as well as rigorous validation and verification. Furthermore, good tool support is essential to support the industrial application of these design techniques.

In 2005 the RODIN (*Rigorous Open Development Environment for Complex Systems*) project organised a workshop on *Rigorous Engineering of Fault Tolerant Systems*. REFT 2005[1] was held in conjunction with the Formal Methods 2005 conference at Newcastle University. The aim of this workshop was to bring together researchers who were interested in the application of rigorous design techniques to the development of fault tolerant software based systems.

Such was the success of that event that the organisers decided to prepare a book on the same combination of topics by inviting the authors of the best workshop papers to expand their work and a number of well-established researchers working in the area to write invited chapters. This book contains the refereed and revised papers that came in response. Twelve of the papers are reworked from the workshop; nine of them are totally new. We have also included two provocatively different position statements from Abrial and Amey on the role of programming languages.

The organisers would like to thank the reviewers (some of whom work on RODIN, others are from outside the project): Jean-Raymond Abrial, Elisabeth Ball, Fernando Castor Filho, Patrice Chalin, Ernie Cohen, Joey Coleman, Neil Evans, Massimo Felici, Stefania Gnesi, Stefan Hallerstede, Michael Hansen, Ian Hayes, Alexei Iliasov, Dubravka Ilić, Maciej Koutny, Linas Laibinis, Annabelle McIver, Qaisar Ahmad Malik, César Muñoz, Simin Nadjm-Tehrani, Apostolos Niaouris, Ian Oliver, Patrizio Pelliccione, Mike Poppleton, Shamim Ripon, Colin Snook and Divakar Yadav.

[1] The proceedings are at http://www.cs.ncl.ac.uk/research/pubs/trs/papers/915.pdf

We should particularly like to thank José-Luis Ferández-Villacañas Martin who both gave his time to update the meeting on IST plans and has kindly contributed the Foreword to this volume; and Louise Talbot who has quietly and efficiently handled the collation of this book.

Both in organising REFT 2005 and in publishing this edited book we are aiming to build a network of researchers from the wider community to promote integration of dependability and formal methods research. It is encouraging to see that many of the papers address software based systems that impact peoples' everyday lives such as communications systems, mobile services, control systems, medical devices and business transactions. We hope that you enjoy reading this volume and encourage you to contribute to our aim of closer collaboration between dependability and formal methods research. We expect to organise another event in London in July 2007: details will appear on the project WWW site http://www.cs.ncl.ac.uk/research/projects/detail.php?id=219

August 2006

Michael Butler
Cliff Jones
Alexander Romanovsky
Elena Troubitsyna

Lecture Notes in Computer Science

For information about Vols. 1–4208

please contact your bookseller or Springer

Vol. 4313: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods. IX, 197 pages. 2006.

Vol. 4300: Y.Q. Shi (Ed.), Transactions on Data Hiding and Multimedia Security I. IX, 139 pages. 2006.

Vol. 4292: G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, A. Nefian, G. Meenakshisundaram, V. Pascucci, J. Zara, J. Molineros, H. Theisel, T. Malzbender (Eds.), Advances in Visual Computing, Part II. XXXII, 906 pages. 2006.

Vol. 4291: G. Bebis, R. Boyle, B. Parvin, D. Koracin, P. Remagnino, A. Nefian, G. Meenakshisundaram, V. Pascucci, J. Zara, J. Molineros, H. Theisel, T. Malzbender (Eds.), Advances in Visual Computing, Part I. XXXI, 916 pages. 2006.

Vol. 4290: M. van Steen, M. Henning (Eds.), Middleware 2006. XIII, 425 pages. 2006.

Vol. 4283: Y.Q. Shi, B. Jeon (Eds.), Digital Watermarking. XII, 474 pages. 2006.

Vol. 4281: K. Barkaoui, A. Cavalcanti, A. Cerone (Eds.), Theoretical Aspects of Computing - ICTAC. XV, 371 pages. 2006.

Vol. 4280: A.K. Datta, M. Gradinariu (Eds.), Stabilization, Safety, and Security of Distributed Systems. XVII, 590 pages. 2006.

Vol. 4279: N. Kobayashi (Ed.), Programming Languages and Systems. XI, 423 pages. 2006.

Vol. 4278: R. Meersman, Z. Tari, P. Herrero (Eds.), On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, Part II. XLV, 1004 pages. 2006.

Vol. 4277: R. Meersman, Z. Tari, P. Herrero (Eds.), On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops, Part I. XLV, 1009 pages. 2006.

Vol. 4276: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Part II. XXXII, 752 pages. 2006.

Vol. 4275: R. Meersman, Z. Tari (Eds.), On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, Part I. XXXI, 1115 pages. 2006.

Vol. 4273: I.F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, L. Aroyo (Eds.), The Semantic Web - ISWC 2006. XXIV, 1001 pages. 2006.

Vol. 4272: P. Havinga, M. Lijding, N. Meratnia, M. Wegdam (Eds.), Smart Sensing and Context. XI, 267 pages. 2006.

Vol. 4271: F.V. Fomin (Ed.), Graph-Theoretic Concepts in Computer Science. XIII, 358 pages. 2006.

Vol. 4270: H. Zha, Z. Pan, H. Thwaites, A.C. Addison, M. Forte (Eds.), Interactive Technologies and Sociotechnical Systems. XVI, 547 pages. 2006.

Vol. 4269: R. State, S. van der Meer, D. O'Sullivan, T. Pfeifer (Eds.), Large Scale Management of Distributed Systems. XIII, 282 pages. 2006.

Vol. 4268: G. Parr, D. Malone, M. Ó Foghlú (Eds.), Autonomic Principles of IP Operations and Management. XIII, 237 pages. 2006.

Vol. 4267: A. Helmy, B. Jennings, L. Murphy, T. Pfeifer (Eds.), Autonomic Management of Mobile Multimedia Services. XIII, 257 pages. 2006.

Vol. 4266: H. Yoshiura, K. Sakurai, K. Rannenberg, Y. Murayama, S. Kawamura (Eds.), Advances in Information and Computer Security. XIII, 438 pages. 2006.

Vol. 4265: L. Todorovski, N. Lavrač, K.P. Jantke (Eds.), Discovery Science. XIV, 384 pages. 2006. (Sublibrary LNAI).

Vol. 4264: J.L. Balcázar, P.M. Long, F. Stephan (Eds.), Algorithmic Learning Theory. XIII, 393 pages. 2006. (Sublibrary LNAI).

Vol. 4263: A. Levi, E. Savas, H. Yenigün, S. Balcisoy, Y. Saygin (Eds.), Computer and Information Sciences – ISCIS 2006. XXIII, 1084 pages. 2006.

Vol. 4261: Y. Zhuang, S. Yang, Y. Rui, Q. He (Eds.), Advances in Multimedia Information Processing - PCM 2006. XXII, 1040 pages. 2006.

Vol. 4260: Z. Liu, J. He (Eds.), Formal Methods and Software Engineering. XII, 778 pages. 2006.

Vol. 4259: S. Greco, Y. Hata, S. Hirano, M. Inuiguchi, S. Miyamoto, H.S. Nguyen, R. Słowiński (Eds.), Rough Sets and Current Trends in Computing. XXII, 951 pages. 2006. (Sublibrary LNAI).

Vol. 4257: I. Richardson, P. Runeson, R. Messnarz (Eds.), Software Process Improvement. XI, 219 pages. 2006.

Vol. 4256: L. Feng, G. Wang, C. Zeng, R. Huang (Eds.), Web Information Systems – WISE 2006 Workshops. XIV, 320 pages. 2006.

Vol. 4255: K. Aberer, Z. Peng, E.A. Rundensteiner, Y. Zhang, X. Li (Eds.), Web Information Systems – WISE 2006. XIV, 563 pages. 2006.

Vol. 4254: T. Grust, H. Höpfner, A. Illarramendi, S. Jablonski, M. Mesiti, S. Müller, P.-L. Patranjan, K.-U. Sattler, M. Spiliopoulou (Eds.), Current Trends in Database Technology – EDBT 2006. XXXI, 932 pages. 2006.

Vol. 4253: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part III. XXXII, 1301 pages. 2006. (Sublibrary LNAI).

Vol. 4252: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part II. XXXIII, 1335 pages. 2006. (Sublibrary LNAI).

Vol. 4251: B. Gabrys, R.J. Howlett, L.C. Jain (Eds.), Knowledge-Based Intelligent Information and Engineering Systems, Part I. LXVI, 1297 pages. 2006. (Sublibrary LNAI).

Vol. 4249: L. Goubin, M. Matsui (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2006. XII, 462 pages. 2006.

Vol. 4248: S. Staab, V. Svátek (Eds.), Managing Knowledge in a World of Networks. XIV, 400 pages. 2006. (Sublibrary LNAI).

Vol. 4247: T.-D. Wang, X. Li, S.-H. Chen, X. Wang, H. Abbass, H. Iba, G. Chen, X. Yao (Eds.), Simulated Evolution and Learning. XXI, 940 pages. 2006.

Vol. 4246: M. Hermann, A. Voronkov (Eds.), Logic for Programming, Artificial Intelligence, and Reasoning. XIII, 588 pages. 2006. (Sublibrary LNAI).

Vol. 4245: A. Kuba, L.G. Nyúl, K. Palágyi (Eds.), Discrete Geometry for Computer Imagery. XIII, 688 pages. 2006.

Vol. 4244: S. Spaccapietra (Ed.), Journal on Data Semantics VII. XI, 267 pages. 2006.

Vol. 4243: T. Yakhno, E.J. Neuhold (Eds.), Advances in Information Systems. XIII, 420 pages. 2006.

Vol. 4242: A. Rashid, M. Aksit (Eds.), Transactions on Aspect-Oriented Software Development II. IX, 289 pages. 2006.

Vol. 4241: R.R. Beichel, M. Sonka (Eds.), Computer Vision Approaches to Medical Image Analysis. XI, 262 pages. 2006.

Vol. 4239: H.Y. Youn, M. Kim, H. Morikawa (Eds.), Ubiquitous Computing Systems. XVI, 548 pages. 2006.

Vol. 4238: Y.-T. Kim, M. Takano (Eds.), Management of Convergence Networks and Services. XVIII, 605 pages. 2006.

Vol. 4237: H. Leitold, E. Markatos (Eds.), Communications and Multimedia Security. XII, 253 pages. 2006.

Vol. 4236: L. Breveglieri, I. Koren, D. Naccache, J.-P. Seifert (Eds.), Fault Diagnosis and Tolerance in Cryptography. XIII, 253 pages. 2006.

Vol. 4234: I. King, J. Wang, L. Chan, D. Wang (Eds.), Neural Information Processing, Part III. XXII, 1227 pages. 2006.

Vol. 4233: I. King, J. Wang, L. Chan, D. Wang (Eds.), Neural Information Processing, Part II. XXII, 1203 pages. 2006.

Vol. 4232: I. King, J. Wang, L. Chan, D. Wang (Eds.), Neural Information Processing, Part I. XLVI, 1153 pages. 2006.

Vol. 4231: J. F. Roddick, R. Benjamins, S. Si-Saïd Cherfi, R. Chiang, C. Claramunt, R. Elmasri, F. Grandi, H. Han, M. Hepp, M. Hepp, M. Lytras, V.B. Mišić, G. Poels, I.-Y. Song, J. Trujillo, C. Vangenot (Eds.), Advances in Conceptual Modeling - Theory and Practice. XXII, 456 pages. 2006.

Vol. 4230: C. Priami, A. Ingólfsdóttir, B. Mishra, H.R. Nielson (Eds.), Transactions on Computational Systems Biology VII. VII, 185 pages. 2006. (Sublibrary LNBI).

Vol. 4229: E. Najm, J.F. Pradat-Peyre, V.V. Donzeau-Gouge (Eds.), Formal Techniques for Networked and Distributed Systems - FORTE 2006. X, 486 pages. 2006.

Vol. 4228: D.E. Lightfoot, C.A. Szyperski (Eds.), Modular Programming Languages. X, 415 pages. 2006.

Vol. 4227: W. Nejdl, K. Tochtermann (Eds.), Innovative Approaches for Learning and Knowledge Sharing. XVII, 721 pages. 2006.

Vol. 4226: R.T. Mittermeir (Ed.), Informatics Education – The Bridge between Using and Understanding Computers. XVII, 319 pages. 2006.

Vol. 4225: J.F. Martínez-Trinidad, J.A. Carrasco Ochoa, J. Kittler (Eds.), Progress in Pattern Recognition, Image Analysis and Applications. XIX, 995 pages. 2006.

Vol. 4224: E. Corchado, H. Yin, V. Botti, C. Fyfe (Eds.), Intelligent Data Engineering and Automated Learning – IDEAL 2006. XXVII, 1447 pages. 2006.

Vol. 4223: L. Wang, L. Jiao, G. Shi, X. Li, J. Liu (Eds.), Fuzzy Systems and Knowledge Discovery. XXVIII, 1335 pages. 2006. (Sublibrary LNAI).

Vol. 4222: L. Jiao, L. Wang, X. Gao, J. Liu, F. Wu (Eds.), Advances in Natural Computation, Part II. XLII, 998 pages. 2006.

Vol. 4221: L. Jiao, L. Wang, X. Gao, J. Liu, F. Wu (Eds.), Advances in Natural Computation, Part I. XLI, 992 pages. 2006.

Vol. 4220: C. Priami, G. Plotkin (Eds.), Transactions on Computational Systems Biology VI. IX, 247 pages. 2006. (Sublibrary LNBI).

Vol. 4219: D. Zamboni, C. Kruegel (Eds.), Recent Advances in Intrusion Detection. XII, 331 pages. 2006.

Vol. 4218: S. Graf, W. Zhang (Eds.), Automated Technology for Verification and Analysis. XIV, 540 pages. 2006.

Vol. 4217: P. Cuenca, L. Orozco-Barbosa (Eds.), Personal Wireless Communications. XV, 532 pages. 2006.

Vol. 4216: M.R. Berthold, R. Glen, I. Fischer (Eds.), Computational Life Sciences II. XIII, 269 pages. 2006. (Sublibrary LNBI).

Vol. 4215: D.W. Embley, A. Olivé, S. Ram (Eds.), Conceptual Modeling - ER 2006. XVI, 590 pages. 2006.

Vol. 4213: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Knowledge Discovery in Databases: PKDD 2006. XXII, 660 pages. 2006. (Sublibrary LNAI).

Vol. 4212: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), Machine Learning: ECML 2006. XXIII, 851 pages. 2006. (Sublibrary LNAI).

Vol. 4211: P. Vogt, Y. Sugita, E. Tuci, C. Nehaniv (Eds.), Symbol Grounding and Beyond. VIII, 237 pages. 2006. (Sublibrary LNAI).

Vol. 4210: C. Priami (Ed.), Computational Methods in Systems Biology. X, 323 pages. 2006. (Sublibrary LNBI).

Vol. 4209: F. Crestani, P. Ferragina, M. Sanderson (Eds.), String Processing and Information Retrieval. XIV, 367 pages. 2006.

¥427.00元

Table of Contents

Position Papers

Train Systems

Jean-Raymond Abrial

ETH Zurich, Switzerland
jabrial@inf.ethz.ch

Abstract. This chapter presents the modelling of a software controller in charge of managing the movements of trains on a track network. Some methodological aspects of this development are emphasized: the preliminary informal presentation of the requirements, the careful definition of a refinement strategy, the attention payed to the precise mathematical definition of the train network, and the modelling of a complete system including the external environment. A special attention is given to the prevention of errors and also (but to a less extend) to their tolerance. The modelling notation which is used in this presentation is Event-B.

Keywords: Event-B, Requirement, Refinement, Failure, Correct Construction.

1 Informal Introduction

The purpose of this chapter[1] is to show the specification and construction of a complete computerized system. The example we are interested in is called a *train system*. By this, we mean a system that is practically managed by a *train agent*, whose role is to control the various trains crossing part of a certain *track network* situated under his supervision. The computerized system we want to construct is supposed to help the train agent in doing this task.

Before entering in the informal description of this system (followed by its formal construction), it might be useful to explain the reason why we think it is important to present such a case study in great details. There are at least four reasons which are the following:

1. This example presents an interesting case of quite complex data structures (the track network) whose mathematical properties have to be defined with great care: we want to show that this is possible.
2. This example also shows a very interesting case where the reliability of the final product is absolutely fundamental: several trains have to be able to safely cross the network under the complete automatic guidance of the software product we want to construct. For this reason, it will be important to study the bad incidents that could happen and which we want to either completely avoid or safely manage. In this chapter however, we are more concerned by *fault prevention* than *fault tolerance*. We shall come back to this in the conclusion.

[1] This work has been partly supported by IST FP6 Rigorous Open Development Environment for Complex Systems (RODIN, IST-511599) Project.

M. Butler et al. (Eds.): Fault-Tolerant Systems, LNCS 4157, pp. 1–36, 2006.

3. The software must take account of the external environment which is to be carefully controlled. As a consequence, the formal modelling we propose here will contain not only a model of the future software we want to construct but also a detailed model of its environment. Our ultimate goal is to have the software working in perfect synchronization with the external equipment, namely the track circuits, the points, the signals, and also the train drivers. We want to *prove* that trains obeying the signals, set by the software controller, and then (blindly) circulating on the tracks whose points have been positioned, again by the software controller, that these trains will do so in a completely safe manner.
4. Together with this study, the reader will be able to understand the kind of methodology we recommend. It should be described, we hope, in sufficiently general terms so that he or she will be able to use this approach in similar examples.

We now proceed with the informal description of this train system together with its informal (but very precise) definitions and requirements. We first define a typical track network which we shall use as a running example throughout the chapter. We then study the two main components of tracks, namely points and crossings. The important concepts of blocks, routes, and signals are then presented together with their main properties. The central notions of route and block reservations are proposed. Safety conditions are then studied.This is followed by the complementary train moving conditions allowing several trains to be present in the network at the same time. We propose a number of assumptions about the way trains behave. Finally we present possible failures that could happen and the way such problems are solved.

The formal development (model construction) is preceded by the *refinement strategy* we shall adopt in order to proceed in a gentle and structured manner. This is followed by the formal model construction.

1.1 Methodological Conventions for the Informal Presentation

In the following sections, we give an informal description of this train system, and, together with this description, we state what its main *definitions and requirements* are. Such definitions and requirements will be inserted as separate labelled boxes in the middle of an explanatory text. These boxes must all together clearly define what is to be taken into account by people doing the formal development. The various definitions and requirements will be labelled according to the following taxonomy:

ENV	Environment
FUN	Functional
SAF	Safety

MVT	Movement
TRN	Train
FLR	Failure

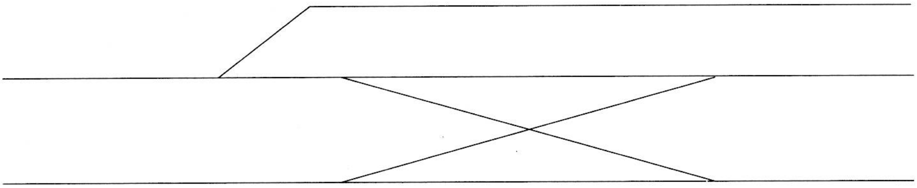
- "Environment" definitions and requirements are concerned with the structure of the track network and its components.
- "Functional" definitions and requirements are dealing with the main functions of the system.
- "Safety" definitions and requirements define the properties ensuring that no classical accidents could happen.
- "Movement" definitions and requirements ensure that a large number of train may cross the network at the same time.
- "Train" definitions and requirements define the implicit assumptions about the behavior of trains.
- "Failure" definitions and requirements finally define the various failures against which the system is able to react without incidents.

Here is our first very general requirement:

The goal of the train system is to safely control trains moving on a track network	FUN-1

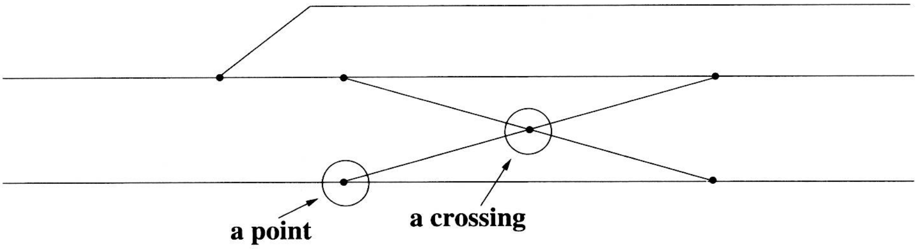
1.2 Network Associated with a Controlling Agent

Here is a typical track network that a train agent is able to control. In what follows, we are going to use that network as a running example:



1.3 Special Components of a Network: Points and Crossings

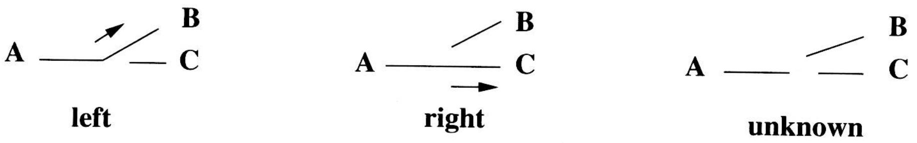
Such a network contains a number of *special components*: these are the *points* and the *crossings* as illustrated in the following figure (five points and one crossing).



A point is a device allowing a track to split in two distinct directions. A crossing, as its name indicates, is a device that makes two different tracks crossing each other. In what follows we briefly describe points and crossings.

A train network may contain some special components: points and crossings	ENV-1

Point. A point special component can be in three different positions: left, right, or unknown. This is indicated in the following figure.



left right unknown

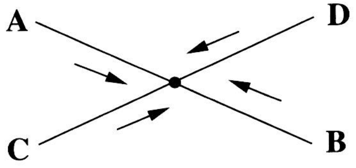
Note that the orientation from A to C is sometimes called the *direct track* whereas the one from A to B is called the *diverted track.* In what follows however, we shall continue to call them right and left respectively are there is no ambiguity in doing so.

In the first two cases above, the arrow in the figure shows the convention we shall use to indicate the orientation of the point. Note that these arrows do not indicate the direction followed by a train. For example, in the first case, it is said that a train coming from **A** will turn left, a train coming from **B** will turn right, and a train coming from **C** *will probably have some troubles*! Also note that a train encountering a point oriented in an unknown direction (third case) might have some trouble too, even more if a point suddenly changes position while a train is on it (we shall come to this in section 1.8).

The last case is the one that holds when the point is moving from left to right or vice-versa. This is because this movement is supposed to take some time: it is performed by means of a motor which is part of the point. When the point has reached its final position (left or right) it is locked, whereas when it is moving it is unlocked. Note however that in the coming development we shall not take this into account. In other words, we shall suppose, as a simplification, that a *point moves instantaneously and that it is thus always locked.* In other words, the unknown case is not treated, we then just require in this development that a point may have only two positions: left or right.

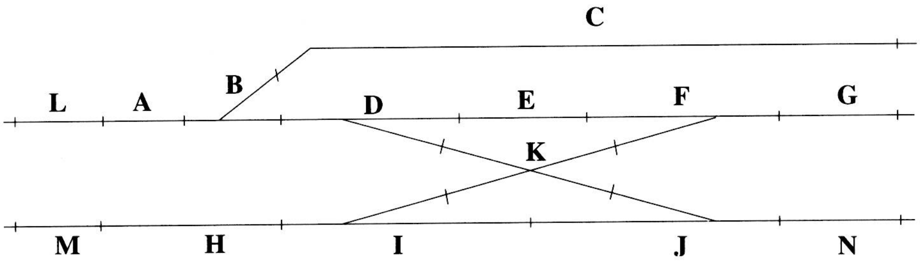
A point may have two positions: left or right	ENV-2

Crossing. A crossing special component is completely static: it has no state as points have. The way a crossing behaves is illustrated in the following figure: trains can go from **A** to **B** and vice-versa, and from **C** to **D** and vice-versa.



1.4 The Concept of Block

The controlled network is statically divided into a fixed number of *named blocks* as indicated in the following figure where we have 14 blocks named by single letters from A to N:



A track network is made of a number of fixed blocks	ENV-3

Each block may contain at most one special component (points or crossings).

A special component (points or crossings) is always attached to a given block. And a block contains at most one special component	ENV-4

For example in our case, block C does not contain any special component, whereas block D contains one point, and block K contains a crossing. Each block is equipped with a, so-called, *track circuit* which is able to detect the presence of a train on it. A block can thus be in two distinct states: unoccupied (no train on it) or occupied (a train is on it).

A block may be occupied or unoccupied by a train	ENV-5

In the following figure, you can see that a train is occupying the two adjacent blocks D and K (this is indicated in the figure by the fact that the blocks in question are emphasized).