URL: http://www.elsevier.nl/locate/entcs/volume82.html 11 pages

Modelling and Validating a multiple-configuration railway signalling system using SDL

A. Fantechi [1,2] E. Spinicci [1,3]

Dipartimento di Sistemi e Informatica
Università degli Studi di Firenze
Firenze, Italy

Abstract

This paper discusses some issues about the usage of SDL and related commercial SDL support tools for the validation of a railway signalling system: in particular, the issue of the multiple configurations presented by this system is addressed, discussing the possible strategies to validate the system regardless to the actual configuration.

1 Introduction

Formal methods have been recognized as a mean for preventing the introduction of software faults in a safety critical system. This is particularly true in the field of railway signalling systems, so that even CENELEC guidelines [4] for software development recommend the use of formal methods. Several success stories have been reported in this field, starting from the early application of B in the development of the control software for the Paris metro [6, 7].
Nowadays, the industrial trend is directed to the adoption of formal verification techniques to validate the design, integrating them within the existing development process: industries seem more keen to accept formal verification techniques assessing the quality attributes of their products, obtained by a traditional life cycle, rather than a fully formal life cycle development, due to the lower training and innovation costs of the former.
The availability of commercial tools for the support of formal specification

[1] This work has been partially supported by the Italian Ministry of University and Research within the COFIN 2001 project "Quack: a platform for the quality of new generation integrated embedded systems"
[2] Email: fantechi@dsi.unifi.it
[3] Email: spinicci@dsi.unifi.it

and verification has boosted in the recent years the use of those specific formalisms that the tools support. Two major examples in this direction are SDL [1], a standard developed within the telecommunication industry, and the Statecharts supported by the iLogix Statemate tool. We refer to [8] and [5] for two notable examples of application of such formalisms to railway signalling systems.
In a collaboration with General Electric Transportation Systems we have conducted a pilot project aimed at the evaluation of the introduction of the SDL technology in their development cycle. The project, described in [3], has addressed the specification in SDL of a railway signalling system, and its simulation. In this paper we present some further discussion on the validation made possible by some of the adopted SDL support tools, especially addressing the issues raised by the multiple configuration nature of the system at hand.

Indeed, we can describe the system as a distributed system, which is statically configured by means of a varying number of general components connected under a given topology. Each component can be configured according to the way they are connected each other, by means of suitable configuration parameters.

We have therefore planned a validation campaign made at two following levels:

- validating the single units composing the system in all the possible configurations;
- validating typical configurations made by several units.

In this paper, we discuss the approach followed at the first level, by evaluating the most suitable validation techniques for this purpose.

2 Description of the SCA system

The SCA system (Sistema Conta Assi, axel counter system), developed by General Electric Transportation Systems, G.E.T.S., is a device which counts the number of wheelsets belonging to trains which are travelling in a given railway section. This operation is made using suitable sensors, which are placed aside the track. The consecutive calculations of the number of wheelsets entering and leaving the railway section allows to establish if the section itself is free (if the number of leaving wheelsets is equal to the number of entering wheelsets) or is still occupied by a train. This information is used to enable or to forbid the next train to enter the section, by means of a suitable (external) semaphore signalling.

The SCA system is composed by several Control and Acquisition Units, named UCAs, placed along the tracks, and by Detection Points placed aside the track, named PRAs. The UCA-PRA connection is shown in Figure 1. A PRA is formed by two wheel detectors named DRTs, located each at one of the rails, for redundancy. The two detectors must give the same signal

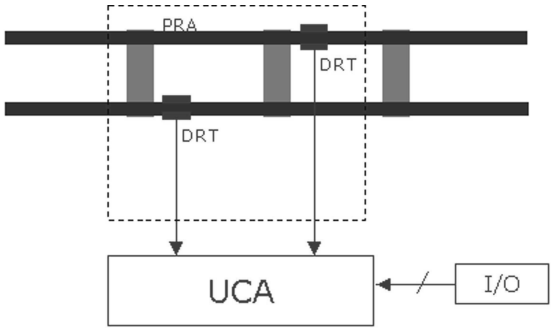

Fig. 1. UCA-PRA connection.

consecutively. If only one of them sends the signal of a passing wheel, the UCA goes in a "fail-safe" state, in which a suitable automatic block relay, connected to the external signalling system, is disabled. A typical effect is to set all signals to red.

UCA units are located at the beginning of a section; they receive and process the signals modulated by the DRTs, in order to determine the number of axels running on the track; these data are sent to remote UCAs via modem. Using remote data, the UCA can determine whether the section is still occupied or not, and in the first case it stops a following train by disabling the automatic block relay. Since a track can be generally run in two directions, UCAs are able to check both the trains entering the railway section and the leaving ones.

2.1 UCA functions and the configurations of SCA system

The SCA type depends on the number of UCAs constituting the system, and on the configuration data of UCAs themselves. Among these we mention the most important configuration parameters:

(i) UCA role in a modem communication (master, slave);

(ii) Number of checked railway sections;

(iii) Number of connected PRAs;

(iv) Type of the SCA system (single-section, multiple-section);

The main UCA functionalities are the following:

- *Core function and Input/Output management.* This is the core axel counting functionality, but extends to read all the input signals (from DRTs or from other UCAs), handled by means of proper filters, and to produce according output, e.g the enabling/disabling of the automatic block relay.
- *Error management.* This functionality analyzes all the errors occurred in the system units. If a *vital error* occurs, the section is declared busy, disabling the automatic block relay. All the error data are saved in a NOVRAM, together with the time and date of occurrence. In this way, error data can be downloaded from a diagnostic PC.

- *Diagnostics.* This function allows a runtime hardware check to be performed. It is possibile to get the diagnostic results, error and configuration data by connecting a PC to the UCA RS232 serial port.

2.1.1 Single-section SCA

A single-section SCA can check only one railway section (Figure 2). It is formed by two UCAs, which are linked via a modem connection, each of them placed at one end of the section; they receive signals coming from PRAs, which can be in a number from 2 to 4.

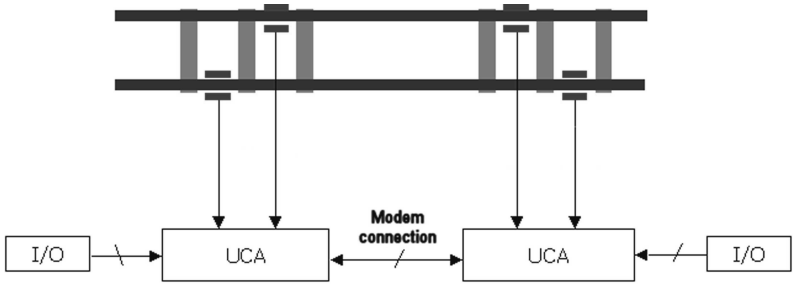

Fig. 2. Single-section SCA.

2.1.2 Multiple-section SCA

A SCA system in multiple-section configuration (Figure 3) can check up to 6 railway sections at the same time: in addition to the header UCA, located at each end of the section, there are one or more intermediate UCAs. Every UCA communicates by means of two different modem channels. Moreover, up to 4 PRAs can be linked to each UCA.

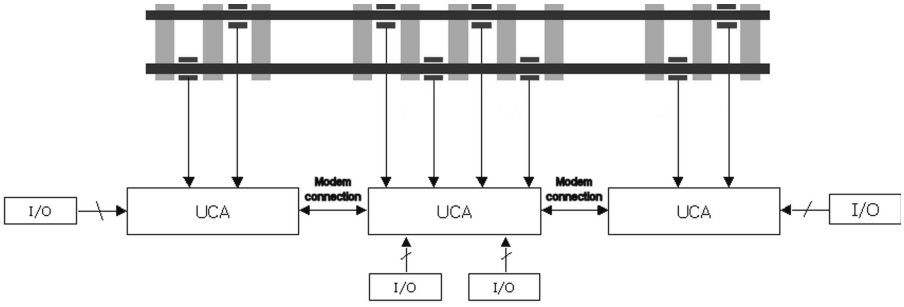

Fig. 3. Multiple-section SCA.

3 Validation strategies and tools

Within the pilot project conducted with General Electric Transportation Systems, a SDL specification of the SCA system was developed using the Cin-

derella SDL tool, exploiting the simulation capabilities of the tool to perform an initial verification of its conformance to the system requirements.

We delay to section 4 some details on the SDL SCA specification, since we want first to introduce the verification activities that we are going to exploit, recalling that the aim of this work is however limited to the analysis of what can be achieved in this direction by commercial SDL support tools.

Validating a system is often conceived as providing the evidence that all possible behaviours of the system have been verified not to produce problems, in reference to the requirement documents. The *coverage* concept is widely used in the area of software testing, at this regard, to measure the percentage of the system which has been exercised by the performed tests: particular structures of a program are chosen as the unit of coverage, so that the notion of *statement coverage, branch coverage, condition coverage,* etc... are commonly used.

In the case of a SDL specification, a *symbol coverage* concept is used: a 100% symbol coverage is achieved by a validation effort if all the graphic elements appearing in the SDL descriptions of the system state machines (that is, in the SDL processes) have been exercised. This assures that all the states defined in the SDL processes have been properly validated, but it may be that not all the possible values of the process variables have been considered: in fact, if we consider the system as composed by extended-finite state machines (EFSMs), two system states are different not only because of different states of a process, but also because of different values assigned to the process variables, even if in presence of the same process state. Symbol coverage hence abstracts from values of variables.

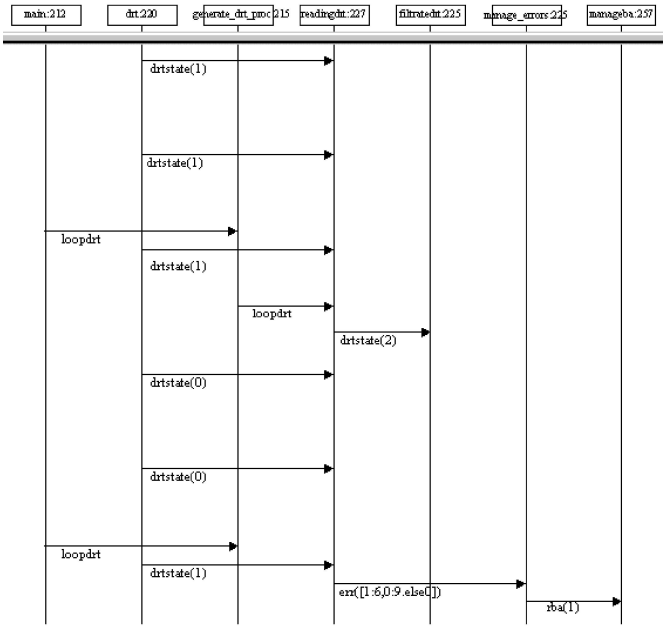

Fig. 4. DRT Buffer reading and writing.

A first approach to validate a SDL specification is by *simulation*. During the simulation activity, the system is handled as a black-box, which, following suitable input stimuli, must send the expected output signals to the system environment. Simulation is essentially a testing performed at the specification level. Several SDL tools, like Cinderella or Telelogic TAU SDT provide support for simulation. If any undesired behavior is revealed, a possible flaw in the system design can be located searching the parts of the model which processed the input signals. Such processing can be analyzed in detail using the Message Sequence Chart formalism [2], as shown in Figure 4, taken from [3], where this MSC is used to describe an anomaly detected during the simulation of the SDL model of SCA.

Unfortunately, simulation can verify only one path in the system state space at a time, and hence a very partial coverage of the model can be obtained. An alternative technique supported by Telelogic TAU SDT is called in SDT terminology "*validation*" and consists in exploring the state space generated by the processes of the system by executing a suitable algorithm: either exhaustive search until reaching a specified depth, or partial search following several criteria in the choice of the next state to be explored. Contrary to simulation, the validator itself manages the signals from the environment which are to be sent to the system, in order to find possible undesired behaviours of the model. The SDT Validator provides a list of situations that may imply an undesired behaviour (such as deadlock, implicit signal consumptions, etc...) that can be checked by the Validator itself. Moreover, MSCs can be used to explicitely define undesired (or desired) behaviours. The notion of symbol coverage is usefully applied also at validation, because real SDL systems often have a very large state space, and hence one is interested to know whether the validation has covered interesting aspects of the system.

4 Modeling UCA multiple-configuration with SDL

4.1 Modeling multiple configurations with SDL

A common approach to modeling a system composed by a variable number of components, number which depends on configuration parameters, is to define some notion of type of components, and to declare as many instances of this type as are needed in the desired configuration. This approach is especially common in object-oriented design.

In SDL, it is possible to define process types, and to dynamically create new instances of a process type whenever they are needed. Therefore, in the model of the SCA system we are going to present, each UCA has a number of instances for each component process, depending on the configuration parameters. The technique used to model such a system is to take advantage of the SDL semantics in generating the desired configuration: a Configura-

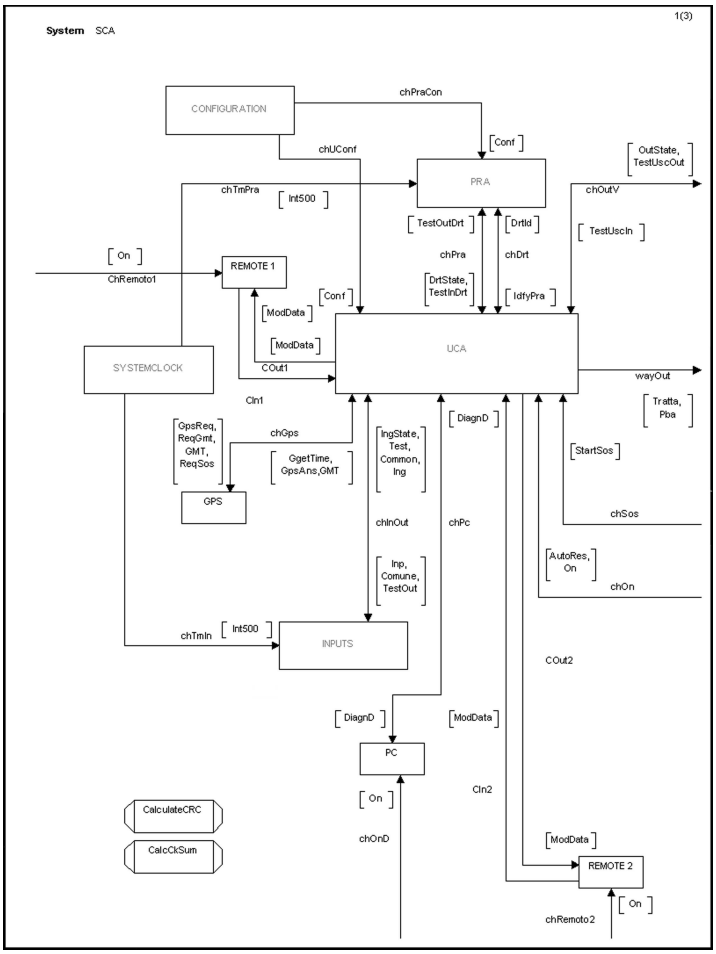

Fig. 5. SCA SDL Formal Specification.

tion block is defined with the aim of handling the parameters describing the configuration: then, the needed instances for each process can be generated by means of a SDL create operation. So, when starting a simulation, a first step of system instantiation occurs, then the model can be verified by sending suitable stimuli, and using only one SDL model: to switch between more configurations, it suffices to change the configuration parameters.

4.2 Modeling the SCA System with SDL

We give now some details about the SDL specification of the SCA system, reported in Figure 5. We can recognize the Configuration block that stores the configuration parameters, and generates at the system startup the number of instances of PRA and diagnostic processes, connected to the block UCA, that are allocated to simulate the desired system configuration. The configuration parameters also identify the role of the UCA block during modem communication (master, slave), depending on the type of the SCA system to be simulated (single-section, multiple-section): for example, if the UCA block represents an intermediate UCA in a multiple-section configuration, the two

blocks REMOTE1 and REMOTE2 simulate the remote UCAs which the UCA block is connected to. Diagnostic communications are specified by means of the PC block, that simulates the external RS232-connected PC.

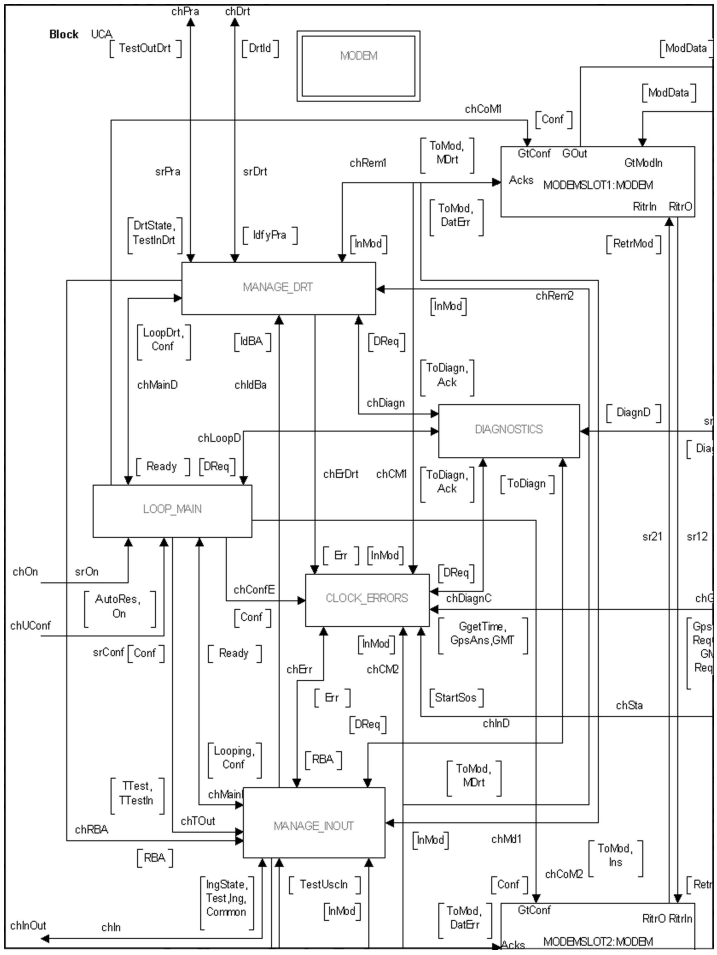

Fig. 6. Substructure UCA in SCA system.

From what we have said, it appears that in Figure 5 are represented not only the UCA(s), but also the other blocks costituting the system, therefore a SDL specification of their behaviour is needed, even though they are not all actually implemented by software components. Their definition is needed to simulate the *environment* in which the UCA, which is the software component of interest, is going to live.

The most interesting unit of this SDL system is therefore the UCA, for which we show the internal structure in Figure 6. The MANAGE_DRT and MANAGE_INOUT blocks handle the signals coming from DRTs and the generation of errors, if the input signals are not the expected ones. The DIAGNOSTICS block concerns the diagnostic requests from the external PC. Communications to remote UCAs are modeled by the two instances MODEMSLOT1 and MODEMSLOT2 of the MODEM block type, which implements the diagnos-

tic communication protocol. Accordingly to safety requirements, system tests are performed by a main loop inside the block LOOP_MAIN. Moreover, the LOOP_MAIN block schedules all the processes.

4.3 Validating multiple configurations

Since with the given specification all possible configurations are modeled within the system structure, in principle "validation" (in SDT terms) is potentially able to address the system under all its possible configurations, by exhaustively exploring the reachable state space under any possible configuration option.
In a model formalized with the previous technique, however, the problem of the state space explosion arises, due to dynamic creation of process instances, even if the ranges of the configuration parameters are given some finite (and in our case, small) bound.
Moreover, we have already said that the exhaustive search nature of validation, aiming at a high symbol coverage, is related to the state space contributed by the processes defined in the SDL specification, and not to the contribution to the state space given by variables. This means that even validation does not explore all possibilities in terms of variables values. This has the effect, in our case, since the choice of different configurations is made early in the system behaviour on the basis of configuration parameters (stored in variables), that some large sections of the model may result uncovered. Actually, some validation tests using Telelogic TAU SDT Validator resulted in a symbol coverage of the model of 37%.

4.4 UCA Incremental Validation using MSCs

To aim for a higher symbol coverage, the idea is to explicitly validate one configuration at a time, by sending from the environment the configuration parameters when running each validation test. Of course, we need to come back to the starting state after each validation is finished: this is possible if we use SDT Navigator to walk back into the state space until the starting state is found: then, the next validation with another configuration can be run. In this way, the symbol coverage of the model is automatically incremented after each validation.
Moreover, in SCA modem and diagnostic protocols, particular message sequences are expected for the transmission of error data to the external PC and to communicate to the remote UCAs the passing of a train through the checked railway section. Since the validator randomly generates the signal sequences sent from the environment to the system, for a large model like SCA the expected sequences are unlikely to be sent. As a result, the model sections in which the correct signal sequences are managed are difficult to cover, and the validation process is quite slow. To improve state space reduction, coverage and speed of the validation process, MSCs representing the

correct sequences of protocol messages are to be used in order to pilot validation tests. In this way, only the messages in the MSC are taken into account by the Validator, thus reducing the state space to be explored. Obviously, this amounts to validate the UCA in presence of a correct environment. It is also important, however, to verify the resilience of UCA in presence of an incorrect behaviour of its environment. Undesired behaviours can be represented with other suitable MSCs. Given assumptions on the occurrence of faults in the environment may be taken into account in the definition of such incorrect behaviours: a typical example are single-failure assumptions. If the system behaves correctly in presence of input sequences according to such assumptions, we can positively conclude on its fault-tolerant design. Using MSC driven validation, we have soon reached a symbol coverage of 52%. Furtherly, refining the validation strategy, we can easily reach higher levels of coverage.

5 Conclusions

We have discussed an experience in the validation of a system specified in SDL by means of commercial support tools, namely the Telelogic TAU SDT Validator and Simulator. In particular we have concentrated on the multiple configuration nature of the system, showing how the tools should be used to best address this issue.
This work has been done as part of a wider effort to evaluate the feasibility of the use of such tools in the development cycle of an industrial product. Next work will address two issues:

- the validation of the whole SCA system, under different configurations
- the automatic generation of code from the SDL specification, and its evaluation in terms of time and space characteristics.

For what concern the first issue, we recall that the current SCA model is constituted by a single UCA which is configured according to the configuration parameters; remote connected UCAs are simulated by REMOTE1 and REMOTE2 blocks, in the environment of the UCA block.
It is therefore possible to substitute such blocks with copies of the UCA itself, and to perform a validation of the whole system. Obviously, the state space will increase and the coverage will decrease accordingly. It is also interesting to notice that Telelogic SDT allows to perform a distributed simulation and/or validation by connecting many SDL models, each of them can be verified on a single computer. Hence a complete distributed verification of the SCA system can be performed by running each constituting UCA model, properly configured, on a different PC.
For what concerns the automatic generation of code from the SDL model, SDT Cmicro is one of the C code generators provided with Telelogic Tau SDT: its main feature lies in the optimized generation of code for embedded systems. Unfortunately, some SDL restrictions occur when using this

code generator: for example, process instances cannot be created by passing parameters with the FPARS SDL statement in the process header, and the multiple-configuration technique used to model the system utilizes exactly this method to create process instances depending on the configuration parameters.
We should therefore resort to include statically all instances of processes in the specification, regardless of the configuration. The result is a less elegant SDL model, which is counterbalanced by the benefits, in terms of time spent in code developing, which can be obtained by automatically generating the code.

References

[1] ITU-T, *Recommendation Z.100 - Specification and Description Language (SDL).* Geneva, 1992

[2] ITU-T, *Recommendation Z.120 - Message Sequence Charts (MSC).* Geneva, 1999.

[3] S. Bacherini, S. Bianchi, L. Capecchi, C. Becheri, A. Felleca, A. Fantechi, E. Spinicci, *Modelling a railway signalling system using SDL.* Submitted for publication.

[4] CENELEC EN 50128, *Railway Applications – Software for Railway Control and Protection Systems.* European Committee for Electrotechnical Standardization, June 1997.

[5] J. Bohn, W. Damm, H. Wittke, J. Klose, A. Moik, *Modeling and Validating Train System Applications Using Statemate and Live Sequence Charts.* Integrated Design and Process Technology, IDPT-2002, United States of America, June 2002.

[6] C. Da Silva, B. Dehbonei, F. Mejia, *Formal Specification in the Development of Industrial Applications: Subway Speed Control System.* Formal Description Techniques, V (C-10), M. Diaz and R. Groz (Eds) Elsevier Science Publishers B, V, (North-Holland), 1993.

[7] B. Dehbonei, F. Mejia, *Formal Methods in the Railways Signalling Industry.* Proceedings of FME '94, LNCS 873, 1994, pp. 26-34.

[8] A. Cimatti, P. Pieraccini, R. Sebastiani, P. Traverso, A. Villafiorita, *Formal specification and validation of a vital protocol.* Proceedings of FM'99, Toulouse, France, September 1999. LNCS 1709, Springer-Verlag.

Acknowledgements

We want to acknowledge M. Banci and M. Becucci for carrying on the validation experiments, and General Electric Transportation Systems for kindly lending the case study.