*Journal of Universal Computer Science, vol. 6, no. 7 (2000), 582-585*
*submitted: 3/7/00, accepted: 18/7/00, appeared: 28/7/00 © Springer Pub. Co.*

# The Light Control Case Study: *A Synopsis*

Egon Börger
(Università di Pisa, Italy
boerger@di.unipi.it)

Reinhard Gotzhein
(Universität Kaiserslautern, Germany
gotzhein@informatik.uni-kl.de)

**Abstract:** In this synopsis, we classify the solutions to the LCCS contained in this Special Issue according to a number of criteria. Furthermore, we provide brief descriptions of the focus of each solution, the major achievements and possible shortcomings. We leave it to the reader to establish a ranking of the different approaches, taking into account that the objectives of the contributions differ from each other and influence in particular the choice of languages, methods, and tools. Therefore, the synopsis is mainly based on information received from the authors, it does not present an in-depth analysis of the solutions. Nevertheless, we hope that this synopsis supports the reader in finding the right access to this Special Issue.

**Key Words:** requirements engineering, formal specification, case study

## 1 Classification of Solutions

The results of the classification are shown in Tables 1 and 2. Solutions are listed in alphabetical order. We classify the solutions to the LCCS according to the following criteria:

- *Coverage*: the degree of completion in terms of specified needs and treated parts of the building
- *Effort*: measured in person days (8 hours of work per day) spent on the case study
- *Means of Presentation*: lists the main description techniques applied in the case study
- *Description Style*: classifies the solutions into operational and property-oriented specifications
- *Modularity*: indicates whether the specification is subdivided into meaningful parts that can be understood independently
- *Traceability*: indicates whether the units of the problem description, i.e., the needs, can be easily associated with the specification units
- *Reusability*: indicates the support of reuse, and the reuse approach
- *Tool Support*: lists the main tools that have been used in the case study
- *Validation*: indicates what has been validated, and by what method
- *Verification*: indicates what has been verified, and by what method

| Solution: Authors [nr] | Coverage | Effort [days] | Means of Presentation | Description Style | Modularity |
|---|---|---|---|---|---|
| Börger, Riccobene, Schmid [1] | all needs entire floor | 14 | Abstract State Machines | operational and property-oriented | yes (rule macros) |
| Groot, Hooman [2] | all user needs the rooms | 20 | PVS notation (higher order logic) | mainly property-oriented | yes (theories) |
| Heitmeyer, Bharadwaj [3] | all needs single office | 12 | SCR notation | operational | yes (tables) |
| Kronenburg, Peper [4] | all needs entire floor | 20-25 | FOREST (real-time TL, OO) | property-oriented | yes (classes) |
| Smith, Fidge [5] | 10 needs single office | 10-15 | timed trace notation | property-oriented | yes (components) |
| Thompson, Whalen, Heimdahl [6] | all needs (no response time) entire floor | n.a. | RSML$^{-e}$ | operational | yes |

Table 1: Classification of solutions (part 1)

| Solution: Authors [nr] | Trace-ability | Reusability | Tool Support | Validation | Verification |
|---|---|---|---|---|---|
| Börger, Riccobene, Schmid [1] | yes | yes (parame-terization) | AsmGofer (execution) | correctness (simulation), completeness | consistency |
| Groot, Hooman [2] | yes | yes (theories) | PVS (type checking, theorem proving) | correctness (review, putative proofs) | correctness (type checks), consistency (interactive) |
| Heitmeyer, Bharadwaj [3] | yes | yes (framework) | SCR* Toolset (editing, analysis, simulation), Salsa (analysis) | correctness (inspection, user scenarios) | consistency (automatic) |
| Kronenburg, Peper [4] | yes | yes (patterns, classes) | xforest (editing, syntax checking, ps- and html-projections) | completeness (traceability), correctness (review) | completeness (manual) |
| Smith, Fidge [5] | yes | yes (incr. development) | - | - | - |
| Thompson, Whalen, Heimdahl [6] | not easily | not addressed | NIMBUS (simulation), prototyping tool | correctness (user scenarios) | correctness (automatic type checks) |

Table 2: Classification of solutions (part 2)

## 2 Description of Solutions

We now provide brief descriptions of each solution, its focus, the major achievements and possible shortcomings (listed in alphabetical order). The descriptions are mainly based on information received from the authors, they do not present an in-depth analysis of the solutions.

**Solution 1**: E. BÖRGER, E. RICCOBENE, J. SCHMID: CAPTURING REQUIREMENTS BY ABSTRACT STATE MACHINES
A complete[1] formal requirement specification using an Abstract State Machine (ASM) model, removing inconsistencies and ambiguities. This specification is refined into a version that is executable with AsmGofer.

The focus of the work has been to provide a rigorous, well structured and traceable specification of the problem description that is executable and compilable to implementation languages such as C++.

**Solution 2**: A. DE GROOT, J. HOOMAN: ANALYZING THE LIGHT CONTROL SYSTEM WITH PVS
A formal requirement specification covering the needs for the rooms of the building has been developed, using PVS notation, a strongly type higher-order logic. The interactive theorem prover PVS has been applied to check for consistency and correctness.

The focus of this work has been to provide a framework for a straightforward translation of informal needs into a formal specification. The objective was also to do this at several levels of abstraction, starting from requirements that are stated purely in terms of observations users can make, and gradually refining it through actuators and sensors. Some ambiguities and conflicts in the informal problem description have been detected and resolved. A possible drawback is the readability of the specification for people unfamiliar with PVS.

**Solution 3**: C. HEITMEYER, R. BHARADWAJ: APPLYING THE SCR REQUIREMENTS METHOD
A specification of the required system behavior in the SCR (Software Cost Reduction) tabular notation has been developed for a single office. This specification identifies the relevant environmental quantities, partitioning them into monitored and controlled quantities; identifies the system modes (to make the specifications concise); specifies the required relation the system must enforce by describing the values of the controlled quantities in response to changes in the monitored quantities. Entry of the SCR requirements specification into the SCR toolset automatically generates code for an executable system prototype. Additionally, some system design activities are reported.

The focus of the work has been on the application of the SCR method and toolset for the specification, analysis, and simulation of the system requirements, system design, and software requirements of a single office. The authors mention that although the timing behaviour of the system has been specified, techniques for analyzing this behaviour are not yet available.

---

1. "Complete" in this context means that all needs of the problem description are addressed, and that the entire floor has been specified.

**Solution 4**: M. KRONENBURG, C. PEPER: APPLICATION OF THE FOREST APPROACH

A complete formal requirement specification has been developed, using the FoReST approach. FOREST provides a notation that integrates real-time temporal logic and object-oriented structuring concepts. Furthermore, the FOREST process model supports incremental requirements engineering and customer feedback, based on a partial translation of formalized needs.

The focus of the work has been on the creation of a precise and intelligible requirement specification, and on the generation of html documents supporting various traceability relations through hyper links. A shortcoming is the current lack of tool support for semantic analyses.

**Solution 5**: G. SMITH, C. FIDGE: INCREMENTAL DEVELOPMENT OF REAL-TIME REQUIREMENTS

A requirement specification has been developed for 10 needs and a single office, using a timed trace notation based on the timed refinement calculus of Mahony.

Focus of the work has been the incremental development of (possibly changing) requirements. The method mixes requirements engineering steps (requirements specification) with formal development (refinement). The solution demonstrates the complete development from high-level needs to all implementation details for a single office. A drawback is the at present limited tool support for the notation. Scaling to the full system is possible, although not very elegant in the notation.

**Solution 6**: J. M. THOMPSON, M. W. WHALEN, M. P. HEIMDAHL: REQUIREMENTS CAPTURE AND EVALUATION IN NIMBUS

Complete formal, executable system and software requirements specifications have been developed, using RSML$^{-e}$ (Requirements State Machine Language without events). Furthermore, interface prototypes to evaluate both specifications have been produced.

The objective has been to demonstrate the effectiveness of specification-based prototyping in the NIMBUS environment, a toolset supporting RSML$^{-e}$. The need for arrays in the notation has been identified as the only real shortcoming of the approach.