Embedded System Specifications Reuse by a Case Based Reasoning Approach

MIROSLAV SVEDA*, RADIMIR VRBA**
*) Faculty of Information Technology, **) Faculty of Electrical Engineering & Communication
Brno University of Technology
61266 Brno, Bozetechova 2
CZECH REPUBLIC

Abstract: - The paper deals with the reuse of behavioral specifications for embedded systems design employing state or timed-state sequences, their closed-form descriptions by finite-state or timed automata, and corresponding formulae of temporal logics. To demonstrate reusing those formal specifications by means of application patterns, the contribution presents two case studies based on two real design projects: (1) petrol pumping station dispenser controller and (2) multiple lift control system. The last part of the paper provides an insight into case-based reasoning support as applied to formal specification reuse of application patterns represented by finite-state and timed automata; moreover, it discusses possible strategies for automated retrieval of similar patterns from the case library that provides a knowledge base supporting an efficient reuse of formal specifications.

Key-Words: - Embedded systems, formal specifications, finite-state and timed automata, case-based reasoning, reuse

1 Introduction

Every real-world embedded system design stems from decisions based on an application domain knowledge that includes facts about some previous design practice. Evidently, such decisions relate to systems architecture components, called in this paper as application patterns, that determine not only a required system behavior but also some presupposed implementation principles. Application patterns should respect those particular solutions that were successful in previous relevant design cases. While focused on the system architecture range that covers more than software components, the application patterns look in many features like object-oriented design concepts such as reusable patterns [3], design patterns [4], and frameworks [8]. To reuse an application pattern, whose implementation usually consists both of software and hardware components, it means to reuse its formal specification, development of which is very expensive and, consequently, worthwhile for reuse. This paper is aimed at behavioral specifications employing state or timed-state sequences, which correspond to the Kripke style semantics of linear, discrete time temporal or real-time logics, and at their closed-form descriptions by finite-state or timed automata [1].

The following two sections provide two case studies, based on implemented design projects, using application patterns that enable to discuss concrete examples of application patterns reusability. Next three sections introduce the principles and initial results of the knowledge-based support for an efficient reuse of formal specifications stemming from case-based reasoning complemented by fitting retrieval techniques.

2 Petrol Dispenser Control System

The first case study pertains to a petrol pumping station dispenser with a distributed, multiple microcomputer counter/controller [13], formerly specified using so called Asynchronous Specification Language [15]. A dispenser controller is interconnected with its environment through an interface with volume meter (input), pump motor (output), main and by-pass valves (outputs) that enable full or throttled flow, release signal (input) generated by cashier, unhooked nozzle detection (input), product's unit price (input), and volume and price displays (outputs).

2.1 Two-Level Structure

The first employed application pattern is *two-level structure* proposed by Xinyao et al. in [16]: the higher level behaves as an event-driven component, and the lower level behaves as a set of real-time interconnected components. The behavior of the higher level component can be described by the following state sequences of a finite-state automaton with states "blocked-idle," "ready," "full fuel," "throttled" and "closed," and with inputs "release," (nozzle) "hung on/off," "close" (the preset or maximal displayable volume achieved),

"throttle" (to slow down the flow to enable exact dosage) and "error":

blocked-idle $\xrightarrow{release}$ ready $\xrightarrow{hung\_off}$ full_fuel $\xrightarrow{hung\_on}$ blocked-idle
blocked-idle $\xrightarrow{release}$ ready $\xrightarrow{hung\_off}$ full_fuel $\xrightarrow{throttle}$ throttled $\xrightarrow{hung\_on}$ $\xrightarrow{hung\_on}$ blocked-idle
blocked-idle $\xrightarrow{release}$ ready $\xrightarrow{hung\_off}$ full_fuel $\xrightarrow{throttle}$ throttled $\xrightarrow{close}$ $\xrightarrow{close}$ closed $\xrightarrow{hung\_on}$ blocked-idle
blocked-idle $\xrightarrow{error}$ blocked-error
blocked-idle $\xrightarrow{release}$ ready $\xrightarrow{error}$ blocked-error
blocked-idle $\xrightarrow{release}$ ready $\xrightarrow{hung\_off}$ full_fuel $\xrightarrow{error}$ blocked-error
blocked-idle $\xrightarrow{release}$ ready $\xrightarrow{hung\_off}$ full_fuel $\xrightarrow{throttle}$ throttled $\xrightarrow{error}$ $\xrightarrow{error}$ blocked-error

The states "full_fuel" and "throttled" appear to be hazardous from the viewpoint of unchecked flow because the motor is on and the liquid is under pressure -- the only nozzle valve controls an issue in this case. Also, the state "ready" tends to be hazardous: when the nozzle is unhooked, the system transfers to the state "full_fuel" with flow enabled. Hence, the accepted fail-stop conception necessitates the detected error management in the form of transition to the state "blocked-error." To initiate such a transition for flow blocking, the error detection in the hazardous states is necessary. On the other hand, the state "blocked-idle" is safe because the input signal "release" can be masked out by the system that, when some failure is detected, performs the internal transition from "blocked-idle" to "blocked-error."

2.2 Incremental Measurement

The volume measurement and flow control represent the main functions of the hazardous states. The next applied application pattern, *incremental measurement*, means the recognition and counting of elementary volumes represented by rectangular impulses, which are generated by a photoelectric pulse generator. The maximal frequency of impulses and a pattern for their recognition depend on electro-magnetic interference characteristics. The lower-level application patterns are in this case *noise-tolerant impulse detector* and *checking reversible counter*. The first one represents a clock-timed impulse-recognition automaton that implements the periodic sampling of its input with values 0 and 1. This automaton with n states recognizes an impulse after n/2 (n>=4) samples with the value 1 followed by n/2 samples with the value 0, possibly interleaved by induced error values, see the following timed-state sequence:

$(0, q_1) \xrightarrow{inp=0} \ldots \xrightarrow{inp=0} (i, q_1) \xrightarrow{inp=1} (i+1, q_2) \xrightarrow{inp=0} \ldots$
$\ldots \xrightarrow{inp=0} (j, q_2) \ldots \xrightarrow{inp=1} (k, q_{n/2+1}) \xrightarrow{inp=1} \ldots$
$\ldots \xrightarrow{inp=1} (m, q_{n-1}) \xrightarrow{inp=0} (m+1, q_n) \xrightarrow{inp=1} \ldots \xrightarrow{inp=1} (n, q_n) \xrightarrow{inp=0/IMP} (n+1, q_1)$

i, j, k, m, n are integers representing discrete time instances:
$0 < i < j < k < m < n$

For the sake of fault-detection requirements, the incremental detector and transfer path are doubled. Consequently, the *second*, identical *noise-tolerant impulse detector* appears necessary.

The subsequent lower-level application pattern used provides *checking reversible counter*, which starts with the value (h + l)/2 and increments or decrements that value according to the "impulse detected" outputs from the first or the second recognition automaton. Overflow or underflow of the pre-set values of h or l indicates an error. Another *counting automaton* that counts the recognized impulses from one of the recognition automata maintains the whole measured volume. The output of the letter automaton refines to two displays with local memories not only for the reason of robustness (they can be compared) but also for functional requirements (double-face stand). To guarantee the overall fault detection capability of the device, it is necessary also to consider checking the counter. This task can be maintained by *I/O watchdog* application pattern that can compare input impulses from the photoelectric pulse generator and the changes of the total value; evidently, the appropriate automaton provides again reversible counting.

2.3 Fault Management

To prevent unregistered flow, the fail-stop conception used appraises as more acceptable the forced blocking of the dispenser with frozen actual data on displays instead of an untrustworthy issue. The application patterns, so far introduced stepwise, cooperate so that they accomplish a consequent application pattern, *fault management based on fail-stop behavior approximation*, in the form of (a) *hazardous state reachability control* and (b) *hazardous state maintenance*. In all safe states ("blocked-idle," "closed," and "blocked-error"), any fuel flow is disabled by power hardware construction; in the same time, the contents of all displays are protected against any change required by possibly erroneous control system. The system is allowed to reach hazardous states ("ready," "full_fuel," and "throttled") when the installed processors successfully have passed start-up checks and interprocessor communication initiation. The *hazardous state maintenance* includes doubled input path check for detected product impulses and *I/O watchdog* check. Hard kernel items such as the nozzle with hydraulic shut-off and mechanical blocking the hooked nozzle eliminate the danger of explosion in the case of uncontrolled petrol flow.

3 Multiple Lift Control System

The second case study deals with a multiple lift control system based on a dedicated multiprocessor architecture

[14], again originally specified using Asynchronous Specification Language [15]. An incremental measurement device for position evaluation, and position and speed control of a lift cabin in a lift shaft can demonstrate reusability. The applied application pattern, *incremental measurement*, means in this case the recognition and counting of rectangular impulses that are generated by an electromagnetic or photoelectric sensor/impulse generator, which is fixed on the bottom of the lift cabin and which passes equidistant position marks while moving along the shaft. That device communicates with its environment through interfaces with impulse generator and drive controller. So, the first input, I, provides the values 0 or 1 that are altered with frequency equivalent to the cabin speed. The second input, D, provides the values "up," "down," or "idle." The output, P, provides the actual absolute position of the cabin in the shaft.

3.1 Two-Level Structure

The next employed application pattern is the *two-level structure*: the higher level behaves as an event-driven component, which behavior is roughly described by the state sequence

initialization $\rightarrow$ position_indication $\rightarrow$ fault_indication

The lower level behaves as a set of real-time interconnected components. The specification of this lower level can be developed by refining the higher level state "position_indication" into three communicating lower level automata: two *noise-tolerant impulse detectors* and one *checking reversible counter*.

3.2 Incremental Measurement

The first automaton models *noise-tolerant impulse detector*, see the following timed-state sequence:

$(0, q_1) \overset{inp=0}{\rightarrow} \ldots \overset{inp=0}{\rightarrow} (i, q_1) \overset{inp=1}{\rightarrow} (i+1, q_2) \overset{inp=0}{\rightarrow} \ldots \overset{inp=0}{\rightarrow} (j, q_2) \ldots$
$\ldots \overset{inp=1}{\rightarrow} (k, q_{n/2+1}) \overset{inp=1}{\rightarrow} \ldots \overset{inp=1}{\rightarrow} (m, q_{n-1}) \overset{inp=0}{\rightarrow} (m+1, q_n) \overset{inp=1}{\rightarrow} \ldots$
$\ldots \overset{inp=1}{\rightarrow} (n, q_n) \overset{inp=0/IMP}{\rightarrow} (n+1, q_1)$
i, j, k, m, n are integers representing discrete time instances:
$0 < i < j < k < m < n$

The information about a detected impulse is sent to the counting automaton that can also access the indication of the cabin movement direction through the input D. For the sake of fault-detection requirements, the impulse generator and the impulse transfer path are doubled. Consequently, a second, identical *noise-tolerant impulse detector* appears necessary. The subsequent application pattern is the *checking reversible counter*, which starts with the value (h + l)/2 and increments or decrements the value according to the "impulse detected" outputs from the first or second recognition automaton. Overflow or underflow of the preset values of h or l indicates an error. This detection process sends a message about a detected impulse and the current direction to the *counting automaton*, which maintains the actual position in the shaft. To check the counter, an *I/O watchdog* application pattern employs again a reversible counter that can compare the impulses from the sensor/impulse generator and the changes of the total value.

3.3 Fault Management

The approach used accomplishes a consequent application pattern, *fault management based on fail-stop behavior approximation*, in the form of (a) *hazardous state reachability control* and (b) *hazardous state maintenance*. In safe states, the lift cabins are fixed at any floors. The system is allowed to reach any hazardous state when all relevant processors have successfully passed the start-up checks of inputs and monitored outputs and of appropriate communication status. The *hazardous state maintenance* includes operational checks and consistency checking for execution processors. To comply with safety-critical conception, all critical inputs and monitored outputs are doubled and compared. When the relevant signals differ, the respective lift is either forced (with the help of a substitute drive if the shaft controller is disconnected) to reach the nearest floor and to stay blocked, or (in the case of maintenance or fire brigade support) its services are partially restricted. The basic safety hard core includes mechanical, emergency brakes.

4 Application Patterns Reuse

The two case studies presented above demonstrate the possibility to reuse effectively substantial parts of the specifications dealing with petrol pumping station technology for a lift control technology project. While both cases belong to embedded control systems, their application domains and their technology principles differ: volume measurement and dosage control seems not too close to position measurement and control. Evidently, the similarity is observable by employment of application patterns.

The reused upper-layer application patterns presented include the automata-based descriptions of incremental measurement, two-level (event-driven/real-time) structure, and fault management stemming from fail-stop behavior approximations. The reused lower-layer application patterns are exemplified by the automata-based descriptions of noise-tolerant impulse detector, checking reversible counter, and I/O watchdog.

Clearly, while all introduced application patterns correspond to design patterns in the above-explained interpretation, the upper-layer application patterns can be related also to frameworks. Moreover, the presented

collection of application patterns creates a base for a pattern language supporting reuse-oriented design process for a subclass of real-time embedded systems.

5 Knowledge-Based Support

Case-based reasoning, see e.g. [10], differs from other rather traditional methods of Artificial Intelligence relying on case history. For a new problem, the case-based reasoning strives for a similar old solution. This old solution is chosen according to the correspondence of a new problem to some old problem that was successfully solved by this approach. Hence, previous significant cases are gathered and saved in a case library. Case-based reasoning stems from remembering a similar situation that worked in past. For software reuse, case-based reasoning utilization has been studied from several viewpoints, see e.g. [6] and [11].

The case-based reasoning method contains (1) elicitation, which means collecting those cases, and (2) implementation, which represents identification of important features for the case description consisting of values of those features. Case library serves as the knowledge base of a case-based reasoning system. The system acquires knowledge from old cases while learning can be achieved accumulating new cases. Solving a new case, the most similar old case is retrieved from the case library. The suggested solution of the new case is generated in conformity with this retrieved old case.

6 Case-Based Reasoning Application

The problem to be solved arises how to measure the similarity of state-based specifications for retrieval. Retrieval schemes proposed in the literature for software component reuse can be classified based upon the technique used to index cases during the search process [2]: (a) classification-based schemes, which include keyword or feature-based controlled vocabularies; (b) structural schemes, which include signature or structural characteristics matching; and (c) behavioral schemes, which seek relevant cases by comparing input and output spaces of components.

The primary approach to the current application includes some equivalents to the component retrieval schemes mentioned above. All of them, i.e. keyword controlled vocabularies belonging to classification schemes, abstract data type signatures belonging to structural schemes, and state-space trajectories belonging to behavioral schemes can provide promising similarity metrics for retrieval. The first alternative means in this case creating a controlled vocabulary of such archetypal temporal logic or real-time temporal logic formulae that represent some key features of the relevant application patterns. The second alternative is based in this context on the algebraic approach employing some process algebras or real-time process algebras corresponding to operational semantics of the relevant temporal logics. Finally, the third alternative denotes for this purpose a quantification of the similarity by some topological characteristics of associated finite automata state-transition graphs, such as the number and placement of loops. The current research task of our group focuses on experiments enabling to compare those alternatives.

6 Related Work

Our approach resembles work [7] that aims at not only specification reuse but also at specification refinement, selection of optimal components, black box and white box software components reuse and adaptation. In contrary to our method, Jilani, Desharnais and Mili use relational specifications. Similarly, Geppert and Roessler [5] apply reuse for SDL specifications. Inspiring for our continuing research can be also the contribution by Justo, Howells and d'Inverno [9], who provide formal framework for software design methodologies using Z.

7 Conclusions

This paper is devoted to the reuse of behavioral specifications for embedded systems design employing state or timed-state sequences, their closed-form descriptions by finite-state or timed automata, and corresponding formulae of temporal logics. It provides two case studies, which are based on implemented design projects, with application patterns that enable to discuss concrete examples of reusability. The contribution introduces principles and initial results of the knowledge-based support for an efficient reuse of formal specifications stemming from case-based reasoning complemented by appropriate retrieval techniques.

Concurrently, the paper informs about case studies and concepts that are available at the opening phase of the research aiming at knowledge-based support for industrial, embedded systems specification and design. Case-based reasoning was successfully utilized by one of the authors and his colleagues previously for another industrial application, see [12]. Therefore, the authors believe that also this higher-level reuse of methodology can improve design processes.

Acknowledgements:
This research has been partly funded by the Czech Ministry of Education in frame of the Research intention

No. MSM 262200022 - Research in microelectronic systems and technologies, and by the Grant Agency of the Czech Republic through the grant GACR 102/02/1032: Embedded Control Systems and their Inter-Communication.

References:

[1] R. Alur and T.A. Henzinger, Logics and Models of Real Time: A Survey, In: *de Bakker, J.W., et al., Real-Time: Theory in Practice*, Springer-Verlag, LNCS 600, 1992, pp. 74-106.

[2] S. Atkinson, Modeling Formal Integrated Component Retrieval, *Proceedings of the Fifth International Conference on Software Reuse*, IEEE Computer Society, Los Alamitos, California, 1998, pp. 337-346.

[3] P. Coad, P. and E.E. Yourdon, *Object-Oriented Analysis.* Yourdon Press, New York, 1990.

[4] E. Gamma and R. Helm and R. Johnson and J. Vlissides, *Design Patterns -- Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[5] B. Geppert and F. Roessler, The SDL Pattern Approach – A Reuse-Driven SDL Design Methodology, *Computer Networks*, Vol. 35, 2001, pp. 627-645.

[6] S. Henninger, An Environment for Reusing Software Processes. *Proceedings of the Fifth International Conference on Software Reuse*, IEEE Computer Society, Los Alamitos, California, 1998, pp. 103-112.

[7] L.L. Jilani and J. Desharnais and A. Mili, Defining and Applying Measures of Distance Between Specifications, *IEEE Transactions on Software Engineering*, Vol. 27, 2001, pp. 673-703.

[8] R.E. Johnson, Frameworks = (Components + Patterns). *Communications of the ACM*, Vol. 40, No. 10, 1997, pp. 39-42.

[9] G.R. Riberio Justo and P. Howells and M. d'Inverno, Formalising High-Performance Systems Methodologies, *Systems Architecture*, Vol. 45, 1999, pp. 441-464.

[10] J. Kolodner, *Case-based Reasoning*, Morgan Kaufmann, San Mateo, CA, USA, 1993.

[11] N. Soundarajan and S. Fridella, Inheritance: From Code Reuse to Reasoning Reuse, *Proceedings of the Fifth International Conference on Software Reuse*, IEEE Computer Society, Los Alamitos, California, 1998, pp. 206-215.

[12] M. Sveda and O. Babka and J. Freeburn, Knowledge Preserving Development: A Case Study. *Proceedings of the Engineering of Computer-Based Systems*, IEEE Computer Society, Los Alamitos, California, 1997, 347-352.

[13] Sveda, M. (1996) Embedded System Design: A Case Study. *Proceedings of the Engineering of Computer-Based Systems*. IEEE Computer Society, Los Alamitos, California, 260-267.

[14] Sveda, M. (1997) An Approach to Safety-Critical Systems Design. In: (Pichler, F., Moreno-Diaz, R.). *Computer Aided Systems Theory*. Springer-Verlag, LNCS 1333, 34-49.

[15] M. Sveda and R. Vrba, Executable Specifications for Distributed Embedded Systems, *IEEE Computer*, Vol. 34, No. 1, 2001, pp. 138-140.

[16] Xinyao, Y., Ji, W., Chaochen, Z., Pandya, P.K. (1994) Formal Design of Hybrid Systems. In: (Langmaack, H., de Roever, W. P., Vytopil, J.) *Formal Techniques in Real-Time and Fault-Tolerant Systems*. Springer-Verlag, LNCS 863, 738-755.