TOWARDS DOMAIN-SPECIFIC FORMAL SPECIFICATION LANGUAGES FOR RAILWAY CONTROL SYSTEMS

Jan Peleska * **Alexander Baer** ** **Anne E. Haxthausen** ***

* *University of Bremen, jp@informatik.uni-bremen.de*
** *INSY GmbH, Berlin, insy_abaer@csi.com*
*** *Denmark Technical University, ah@it.dtu.dk*

Abstract: The authors motivate and illustrate an approach for domain-specific specification languages in the field of small local railway or tramway control systems. Adopting terms and concepts from the application domain, such languages are the ideal means of communication between users, system engineers and control computer specialists. Semantic rigour is provided by a transformation of the domain-specific representation into wide-spectrum formal specification languages. These transformations offer the possibilities of formal verification and testing against formal specifications, as well as automatic generation of executable programs from specifications. The construction of the transformation can be substantially simplified if the class of control systems is formalised in a generic way using design patterns and frameworks.

Keywords: railway control systems, domain-specific formalisms, design patterns, frameworks

1. INTRODUCTION

1.1 *Motivation*

In this contribution the authors argue that domain-specific formalisms are urgently required in the field of safety-critical Railway Control Systems: Wide-spectrum formal or semi-formal specification languages, while having been advocated for several decades and having proved to be valuable tools for verification specialists, have utterly failed as vehicles to communicate ideas between system engineers with expert knowledge in the application domain and the experts for control computer software. As a consequence, informal natural language descriptions and intuitive pictures without well-defined semantics are still the principle communication means between these groups. Our claim is that the main reason for this problem lies in the misconception that application experts should also have expertise in the computer science domain of formal specification languages. Instead, domain-specific languages, using the "natural" concepts of the application domain with a well-defined semantics should be provided for system engineers to be used for the task of system requirements specification. For the task of software-development and verification, these domain-specific specifications may be compiled into architectural and functional descriptions expressed in the standard wide-spectrum formalisms which have already proved to be suitable for the tasks of verification and generation of executable code. To facilitate this transformation process, we encourage the utilisation of domain-specific design patterns or frameworks, so that the transformation process can be at least partially realised as a pattern instantiation. To illustrate and substantiate our hypotheses, we present an example from the field of railway control systems and sketch the features of a formal domain-specific description

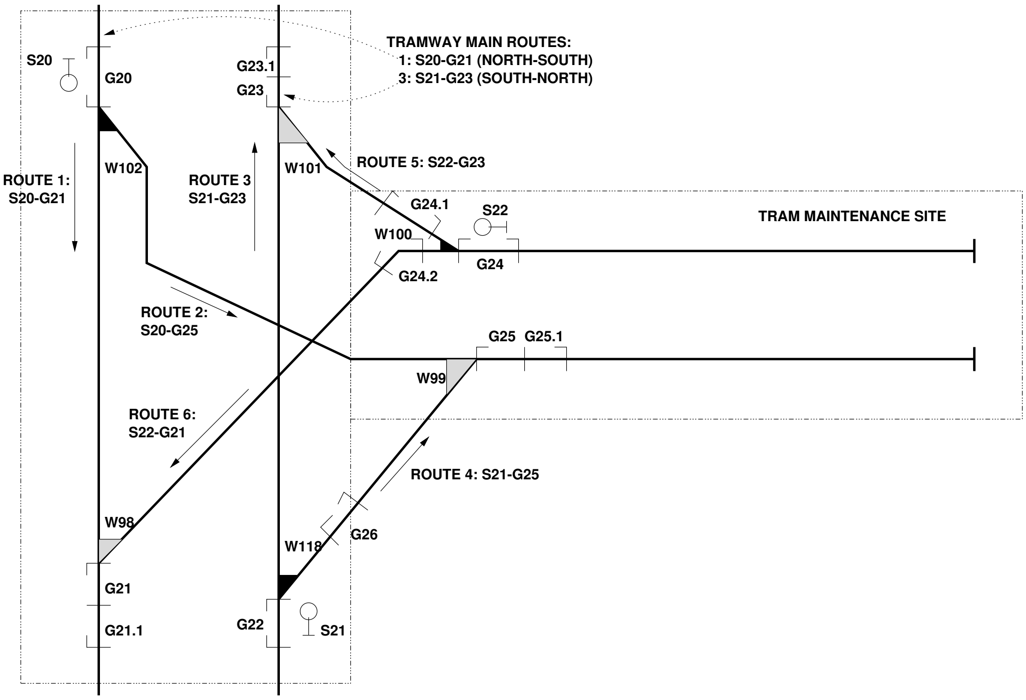

Fig. 1. Tramway network leading from/to a tram maintenance site.

language suitable for the definition of control tasks in small tramway and railway networks.

1.2 *Overview*

The main part of this article is presented in Section 2, where we introduce and illustrate the approach by means of an example from the field of tramway control systems: Starting with a domain-specific description, this leads to an identification of frameworks which – according to our understanding – are essential in the field of train control systems. The formal representation is sketched in the final part of Section 2. Section 3 gives an overview of the benefits which are gained from the approach by facilitating and helping to automate the validation, verification and test (VVT) process, as well as the transformation of the formal descriptions into executable systems. Our results are summarised in Section 4.

2. EXAMPLE: A TRAMWAY CONTROL SYSTEM

In this section, we introduce a tramway control system (TCS) as an example to illustrate our approach. We start with a domain-specific description, which is subsequently transformed into a formal one. The task of the TCS is to control two tramway main routes connected to a maintenance site (see Fig. 1). The details of the TCS requirements will be described below, while introducing the components of a domain-specific description, consisting of the

- tramway network,
- collection of interlocking tables,
- scheduling policy,
- architectural design specification.

2.1 *Specification in the Application Domain*

Tramway Network Specification. The first component of a domain-specific description for railway control systems consists of the *network*, as displayed for our example in Fig. 1. It represents the topology of the network *elements*, that is, the track segments (denoted by G20, G21,...), points (W101, W102,...) and signals (S20, S21, S22) and their positions and interconnections. Typically, the network specification will be kept in a CAD system or any other type of database with visualisation properties that are suitable to display networks. The symbols used in the visualisation are the familiar ones for experts in the railway or tramway domain. The database will store the network specification in such a way that the various elements of the network and the topology can be distinguished and retrieved.

Interlocking Tables. The network specification is complemented by a set of *interlocking tables*, whose purpose it is to fix the possibles routes through the network and the safety conditions which are sufficient to allow trams to pass along these routes. The first of these tables is the *route definition table* below. For each route, it specifies the sequence of track segments and points in the order they are passed by the trams. For example, route 5 specified in the table starts at track segment G24, and a tram taking route 5 passes along point W100 and track segments G24.1, G23 and G23.1. Observe that point W101 is not mentioned in the route definition, since the tramway point technology allows to enter track G23 via W101 from both possible directions without switching W101 into the respective positions.

Route definition table	
Route	**Route Element Sequence**
1	$\langle G20, W102, G21, G21.1\rangle$
2	$\langle G20, W102, G25, G25.1\rangle$
3	$\langle G22, W118, G23, G23.1\rangle$
4	$\langle G22, W118, G26, G25, G25.1\rangle$
5	$\langle G24, W100, G24.1, G23, G23.1\rangle$
6	$\langle G24, W100, G24.2, G21, G21.1\rangle$

For a tram to pass along a specific route, it is necessary to *allocate* the route beforehand. Allocation of a route R involves two activities: First, it must be ensured that no conflicting route is already allocated, second, the points along R must be switched and locked into their proper positions. To this end, the *route conflict table* lists the mutual exclusion conditions for different routes. For example, route 1 may only be allocated to a tram if neither route 2 nor route 6 are allocated at the time route 1 is requested. The exclusion is either ensured by point positions making it impossible for R and the conflicting route to be allocated at the same time (this is marked by "•" in the route conflict table), or by signals switched into halt positions, thereby preventing trams to enter the conflicting route (marked by "○"). Observe that the route conflict table is an auxiliary table which could also be systematically derived from the network specification and the route definition table. However, the conflict table is essential for safety considerations, because it specifies conditions which are necessary to avoid collisions. Therefore it is presented explicitly.

Route conflict table						
Route	**Conflicts with**					
	1	2	3	4	5	6
1		•				○
2	•		○	○		○
3		○		•	○	○
4		○	•			
5			○			•
6	○	○	○		•	

The last interlocking table is also a derived one, but just as essential for the check of safety conditions as the point position table: The *point position table* below specifies for each point which is part of the route whether it has to be locked in straight position (marked by "+") or in branching position ("–"). Readers who are familiar with train control systems may have noticed that the point position table does not guarantee *flank protection*: Points which are not part of a route may be switched into any position. This is allowed for tramway control systems since – at least in Germany – trams drive according to the road traffic regulations.

Point position table					
No.	**FROM**	**TO**	**W100**	**W102**	**W118**
1	S20	G21		+	
2	S20	G25		–	
3	S21	G23			+
4	S21	G25			–
5	S22	G23	–		
6	S22	G21	+		

The *deallocation* of a route is already specified by the route definition table above: A tram has left an allocated route, as soon as it has passed along each element in the sequence associated with the route.

Scheduling Policy. If a tramway control system TCS enforces the conditions specified in the interlocking tables above, this already guarantees safety. However, to avoid deadlock situations in the network and to implement application-specific sequencing of tram routes, a *scheduling policy* may be introduced in order to determine the allocation sequence in presence of conflicting requests. In the configuration described in Fig. 1 for example, it might be useful to give priority to the allocation of routes 1 and 3 in presence of conflicting requests for routes 2,4,5,6: Routes 1 and 3 represent the main tramway lines, while 2,4,5,6 are only needed by trams entering and leaving the maintenance site.

Architectural Design Specification. The specification components introduced so far can be implemented in many ways. Therefore, an architectural design specification is required to fix the boundary conditions for the suitable implementations. In the present example, we assume a distributed architectural design where computers CC_{Tram} inside the trams pass allocation requests via radio transmission to a control computer CC_{Site} on the maintenance site. Computer CC_{Site} keeps the global state information and switches signals and points accordingly.

Due to the usual space limitations in this article, we will focus on the network specification and interlocking tables. An example for a different type of architectural design is given in (Haxthausen and Peleska, 1999).

2.2 *A Framework for Tramway Control Systems*

Recall that *frameworks*, as defined in (E. Gamma and Vlissides, 1995) are generic collections of co-

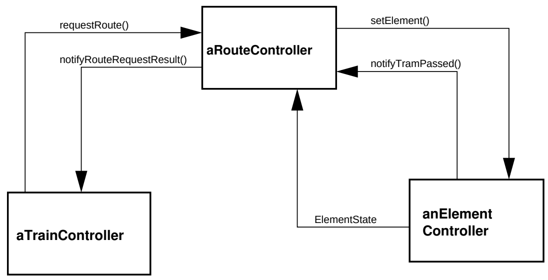

Fig. 2. TCS Object collaboration

operating classes associated with a re-usable design structure. By means of instantiation, subclassing and composition of class instances, frameworks are customised for a specific application context. To elaborate a framework-based generic design and implementation concept for railway/tramway control systems, we suggest to take a triad of distinct abstract controller classes as a starting point:

- The abstract *TrainController* class defines operational interfaces associated with a single tram,
- The abstract *RouteController* class defines operational interfaces related to route allocation and deallocation,
- The abstract *ElementController* class defines operational interfaces related to network elements (track segments, points, signals, tramway crossings).

The collaboration between instances of these classes is sketched in Fig. 2: TrainController objects request route allocations from RouteController objects. To allocate routes, RouteControllers send messages to element controllers to set their local states according to the route definition. ElementController objects notify when a tram has passed the element (this may be implemented by various types of sensors or positioning systems) and reveal their state so that the RouteController object may check whether the element state is correct or whether exception handling becomes necessary.

Observe that at this level of description we are dealing with *abstract* classes and operations: the *setElement()* operation will be differently instantiated, depending on the element type. The usefulness of these abstract operations lies in the fact that the bahavioural patterns of the controller objects may already be defined on this abstract level: concrete classes will introduce various methods for concrete operations, but they can stick to the behavioural pattern (communication and local control structures) already defined on the abstract level.

2.3 *Formal Representation*

Formalising the interlocking tables using mathematical data types, the route definition table *rdt* is a partial mapping from route identifiers to sequences of elements:

$$rdt : ROUTEID \nrightarrow \mathrm{seq}\, ELEMID$$

The route conflict table gives rise to a boolean mapping *rct* with a route r and a set of routes R as arguments, returning *true* iff r conflicts at least with one route in R:

$$rct : ROUTEID \times \mathbb{P}\ ROUTEID \nrightarrow Bool$$

The point position table *ppt* may be abstracted by a mapping from route identifiers and point identifiers to point positions:

$$ppt : ROUTEID \times POINTID \nrightarrow POINTPOS$$

A RouteController object has a state space which may be defined in an abstract way by the set $A : \mathbb{P}\ ROUTEID$ of allocated routes and the set $W : \mathbb{P}(TRAINID \times ROUTEID)$ of pending allocation requests. A simplified version of one RouteController sub-component $RC(W, A)$ is shown below in CSP-style notation: $RC(W, A)$ records requests in W, triggers route allocation for nonconflicting routes in a nondeterministic way and removes de-allocated routes from A. Obviously, the control structure is generic in *rct*.

$$
\begin{array}{l}
RC(W, A) = \\
\quad requestRoute?(tid, rid) \\
\qquad \rightarrow RC(W \cup \{(tid, rid)\}, A) \\
\quad \Box \\
\quad \mathbf{let}\ X = \{(tid, rid) : W \mid \neg\ rct(rid, A)\}\mathbf{in} \\
\quad (\mathbf{if}\ (\emptyset \neq X) \\
\quad\ \mathbf{then}\ (\sqcap (tid, rid) : X \bullet \\
\qquad allocate!(tid, rid) \\
\qquad\quad \rightarrow RC(W - \{(tid, rid)\}, A)) \\
\quad\ \mathbf{else}\ STOP) \\
\quad \Box \\
\quad isAllocated?(tid, rid) \\
\qquad \rightarrow notifyRouteAlloc!(tid, rid) \\
\qquad \rightarrow RC(W, A \cup \{rid\}) \\
\quad \Box \\
\quad isDeallocated?rid \rightarrow RC(W, A - \{rid\})
\end{array}
$$

Another sub-component of RouteController inputs $allocate!(tid, rid)$-requests from $RC(W, A)$ and allocates the route by traversing the sequence $rdt(rid)$, using the point switching commands derived from $ppt(rid)$. After successful allocation, this is signalled via *isAllocated* to $RC(W, A)$. It also controls the de-allocation by recording the *notifyTramPassed()* events und all elements of the route have been passed by the respective trams.

3. EXPLOITING THE APPROACH: AUTOMATED VVT AND TRANSFORMATION INTO EXECUTABLE SYSTEMS

In this section, we give a summary of the benefits resulting from the generic framework-based approach with respect to validation, verification, test (VVT) and transformation into executable systems.

Validation of the Domain Specification. A major advantage of our approach is that it specifies the safety-critical system information in the application domain, where the railway or tramway specialists can take part in the validation process, in order to check the following objectives:

(1) The network specification should be complete and correct.
(2) The route definition table should contain the correct sequences of track segments and points connecting the start and end point of the route as defined in the network specification.
(3) The values defined in the point position table should be consistent with the point positions to be derived for the route from the network specification.
(4) The route conflict table should be minimal and complete.

The basic means for validation is the redundancy provided by the network specification on one hand and the interlocking tables on the other hand. However, even experienced specialists may introduce errors when developing interlocking tables. Therefore it is advisable to apply mechanised evaluation of the validation objectives listed above. Indeed, model checking of the consistency between network specification and interlocking tables is quite easy to mechanise: Objective 2 above is checked by analysing whether each sequence of track segments and points given in the route definition table is a path in the graph representing the tramway network. Objective 3 can be checked on the fly when investigating objective 2. For objective 4 two things have to be investigated: First, non-conflicting routes must not share identical track segments or points. Second, non-conflicting routes must not have segments crossing each other (for example, routes 2,3,6 shown in Fig. 1 possess track segments crossing each other). Therefore, the only remaining validation problem is to ensure the completeness and correctness of the network specification.

Verification of Controller Design. As has been sketched in the previous section, the control structures and communication behaviour of the TCS controller design do not depend on the tramway network: The design is generic with respect to the network-specific interlocking table data. This facilitates the elaboration of a *generic theory* proving that the control software is safe for all possible consistent instances of the interlocking tables. Observe that such a proof requires universal quantification over the generic parameters, that is, the interlocking table values. Therefore the result cannot be established by conventional model checking; instead, one has to proceed using theorem provers or proof checkers. An example of such a generic theory for another type of railway control system has been given in (Haxthausen and Peleska, 1999). The advantage of the generic theory is that it proves design correctness once and for all, so that the recurring VVT-tasks may focus on the validation of the domain specification and of the generated executable system, as described in the following paragraphs.

Generation of Executable Code. Given a verified design it is always a problem to either transform it into executable programming code or perform an *a posteriori* verification of the code against the design specifications. Therefore the authors advocate another approach, using a *real-time interpreter* technique: It is well known that a large variety of untimed and timed specification languages can be semantically modelled using (timed) transition systems, and – in case of finite-state systems – these may be suitably encoded as directed graphs with labelled nodes and edges (the basic transition graph generation technique for specification languages with untimed transition system semantics has been described in (Roscoe, 1994)). A specific graph encoding developed by the first author can be executed in real-time (bounded response time guaranteed) by an associated interpreter reacting to abstract input events and generating abstract actions in response, while preserving the timed transition system semantics encoded in the graph. Transition graph compilers and suitable interpreters exist for Timed CSP (Schneider, 1995) and for a variant of timed automata (Dierks, 1997). The advantage of this approach lies in the fact that the semantic consistency between design specification and executable system has to be established only once for the transition graph compiler and interpreter. Moreover, the interpreters have been developed to ensure optimised real-time behaviour. Experience has shown, that this is extremely hard to achieve for compilers generating programming code from specifications.

Automated Test. Given a generic theory ensuring the safety of the generic design and the interpreter-driven execution technique ensuring the semantic equivalence between design and executable code, the remaining risk consists in an erroneous compiler transforming formal timed automata specifications into binary transition graph representations. Observe that this risk is generally small, since the compiler is re-used for different tramway networks, but nevertheless it cannot be assumed to be completely correct. To address this problem, we suggest to perform *validation tests* of the executable code against the domain specification. The basic idea is to simulate train movements in the specified network and check automatically whether the route allocations which are performed according to the simulated route requests and movements conform to the safety conditions specified in the interlocking tables. This process can be automated with respect to test generation, execution and evaluation, and it is planned to use the RT-Tester tool for this purpose. As a consequence, an arbitrarily high test coverage can be reached without manual interaction, and we claim that the level of confidence which can be achieved with this technique is considerably higher than the confidence to be gained from conventional development and VVT approaches for tramway/railway control systems. This claim is supported by the fact that the automated test strategies used "converge" to refinement proofs with increasing number of test executions (see the RT-Tester information and references to publications listed in (http://www.verified.de/rtt.html, 2000)).

Remaining Risks. The VVT concept sketched above ensures highest dependability for instances of the *TrainController* and *RouteController* classes. We expect, however, that the VVT concept cannot be applied in an analogous way for the *ElementController* instances, since their implementations will generally contain a large amount of hardware device driver software which cannot be suitably handled by timed automata specifications. Therefore we suggest for the *ElementController* instances to perform conventional testing and inspections or – in critical cases – formal code verification. An advantage consists in the fact that we can expect a high degree of re-use for element controllers, since they only depend on the type of peripherals (signals, points etc.) but not on the specific tramway network. Finally, the dependability of communication protocols must be ensured, which is yet another VVT domain of its own. Fortunately, there exists a vast number of proven methods for the development of correct and dependable protocols, so that we recommend to apply the standard methods which are already available.

4. CONCLUSION

In this article, an approach starting from domain-specific description formalisms has been illustrated using an example from the field of tramway control systems. The domain-specific approach offers the possibility to perform important specification and validation steps using the terms and concepts of the domain, so that specialists with domain expertise may contribute to these steps without necessarily being experts in the field of formal methods. Using the example, we have sketched a development and VVT life cycle offering a high degree of automation in both development and VVT activities, thereby reducing the development costs while at the same time increasing the confidence in the final product, since automated VVT results in much higher coverage of VVT objectives than could ever be achieved with convetional methods. The realisation idea is based on the utilisation of generic frameworks describing the different control functions to be implemented on an abstract level.

The verification and validation aspects described in this article are pursued by the first and third author in the context of the HYBRIS (Efficient Specification of Hybrid Systems) project supported by the Deutsche Forschungsgemeinschaft DFG as part of the special programme on Integration of Software Specification Techniques for Scientific Engineering Applications. The aspects of automated executable system generation and testing are investigated by the first and the second author in the R&D project EDASS (Development of data-driven automata for safety-related control systems) funded by the German Ministry of Trade and Commerce.

5. REFERENCES

Dierks, H. (1997). Plc-automata: A new class of implementable real-time automata. In: *Transformation-Based Reactive Systems Development (ARTS'97)* (M. Bertran and T. Rus, Eds.). Vol. 1231. pp. 111–125. Springer LNCS.

E. Gamma, R. Helm, R. Johnson and J. Vlissides (1995). *Design Patterns*. Addison Wesley.

Haxthausen, A. E. and J. Peleska (1999). Formal development and verification of a distributed railway control system. In: *FM'99 – Formal Methods* (J. Woodcock J. M. Wing and J. Davies, Eds.). Vol. 1709. pp. 1546–1563. Springer LNCS.

http://www.verified.de/rtt.html (2000). *Verified's RT-Tester*. Verified Systems International.

Roscoe, A. W. (1994). Model checking csp. In: *A Classical Mind – Essays in Honour of C.A.R. Hoare* (A. W. Roscoe, Ed.). pp. 353–378. Prentice Hall Int.

Schneider, S. (1995). An operational semantics for timed csp. *Information and Computation* **116**, 193–213.