FORMAL VERIFICATION OF SAFETY BEHAVIOURS OF THE OUTDOOR ROBOT RAVON

Martin Proetzsch, Karsten Berns
Robotics Research Lab, University of Kaiserslautern, Germany
proetzsch@informatik.uni-kl.de, berns@informatik.uni-kl.de

T. Schuele, K. Schneider
Reactive Systems Group, University of Kaiserslautern, Germany
schuele@informatik.uni-kl.de, schneider@informatik.uni-kl.de

Keywords: Behaviour-based control, formal verification, outdoor robotics.

Abstract: This paper presents an approach to the formal verification of safety properties of the behaviour-based control network of the mobile outdoor robot RAVON. In particular, we consider behaviours that are used for the computation of the projected vehicle's velocity from obstacle proximity sensor data and inclination information. We describe how this group of behaviours is implemented in the synchronous language Quartz in order to be formally verified using model checking techniques of the Averest verification framework. Moreover, by integrating the automatically generated and verified code into the behaviour network, it can be guaranteed that the robot slows down and stops as required by the given safety specifications.

1 INTRODUCTION

More and more applications like unmanned space travelling, autonomous farming, civil protection, and humanitarian demining require autonomous vehicles navigating in unstructured natural terrain. The diversity as well as missing information for physical models of outdoor scenarios call for a flexible control architecture not requiring a complete knowledge of the environment to achieve robust locomotion. In this context, behaviour-based control networks have proven suitable for appropriate reaction on external influences.

One of the advantages of behaviour-based architectures is emergence: The combination of behaviours leads to proper reactions not directly explainable by the individual components. On the other hand, this feature also poses problems concerning predictability, making it difficult to reason about the correctness of the overall system. However, safety critical applications require proofs that guarantee that given specifications are met. In particular, the correctness of reactive control layers is mandatory for the final behaviour of the robot.

In circuit design, formal verification has already become a standard to avoid design errors. Since all possible input traces of a system are considered by formal verification methods, it is guaranteed that the checked specifications hold under all circumstances. In particular, different kinds of model checking (Schneider, 2003; Schuele and Schneider, 2006) are popular verification methods due to the high degree of automation.

Figure 1: RAVON in rough outdoor terrain.

In this paper, we consider the formal verification of a part of the behaviour-based control network of the mobile outdoor platform RAVON (**R**obust **A**utonomous **V**ehicle for **O**ff-road **N**avigation, see

Fig. 1), namely RAVON's control system that is responsible for slowing down and stopping the vehicle. It is clear that this part is highly safety critical, and therefore, it is very important to guarantee its correctness. To this end, we have implemented this part of the behaviour-based control network in the synchronous programming language Quartz (Schneider, 2001b; Schneider, 2006). We then used the formal verification methods of the Averest framework (Schneider and Schuele, 2005) to check the correctness of our implementation with respect to given safety conditions.

The application of formal methods to verify the correctness of a robot system is not new: In (Diethers et al., 2003), the model checker HyTech was used to analyse a robot program based on skill primitive nets. While HyTech considers hybrid automata as system models to model continuous values of physical properties, our approach is based on discrete transition systems that benefit directly from symbolic model checking techniques. The use of symbolic model checking for the formal verification of a robot system has been reported in (Sharygina et al., 2004). In contrast to our approach, however, the verification is not integrated with code generation.

The use of synchronous languages (Benveniste et al., 2003) for the implementation of safety critical control systems of robots is also not new: In (Sowmya et al., 2002), a controller for a mobile robot (Rug Warrior) has been implemented in the synchronous language Esterel (Berry, 1998). In (Kim and Kang, 2005), the core of the Samsung Home Robot SHR100 has been re-engineered to be verified by means of the Esterel framework. In contrast to our approach, the systems considered in (Sowmya et al., 2002) and (Kim and Kang, 2005) were quite small control programs with only a few control states.

In contrast to the previous work in this area, our approach considers verification as well as (verified) code generation. Moreover, the considered system is not simply a small control program, but a behaviour-based control network with difficult interdependencies.

The outline of the paper is as follows: In the next two sections, we give some details on the outdoor robot RAVON and its behaviour-based control system. In Section 4, we describe the verification framework Averest and give some basics about synchronous languages and model checking. Section 5 contains the results of the verification. Finally, we conclude with a summary and directions for future work.

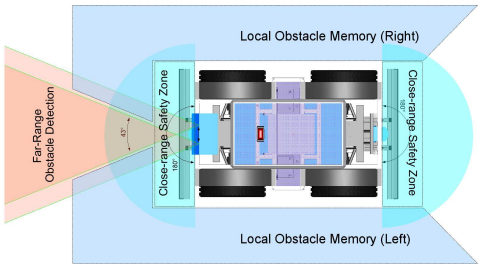

Figure 2: Regions monitored by the obstacle detection and avoidance facilities.

2 THE OUTDOOR ROBOT RAVON

RAVON is a four wheeled off-road vehicle measuring 2.35 m in length and 1.4 m in width and weighting 400 kg. The vehicle features a four wheel drive with independent motors yielding maximal velocities of 3 m/s. In combination with its off-road tires, the vehicle can climb slopes of 100% inclination predestining it for the challenges in rough terrain. Front and rear axis can be steered independently which supports agile advanced driving manoeuvres like double Ackerman and parallel steering.

In order to navigate in a self-dependent fashion, RAVON has been equipped with several sensors. For self localisation purposes, the robot uses its odometry, a custom design inertial measurement unit, a magnetic field sensor, and a DGPS receiver. The sensor data fusion is performed by a Kalman filter (Schmitz et al., 2006) which calculates an estimated pose in three dimensions. Due to gravity measurements of the inertial measurement unit, the control system receives quite precise absolute data for the roll and pitch angle of the vehicle. These are fed into behaviours that are responsible for supervising whether the vehicle might tip over due to critical inclination.

In order to protect the vehicle in respect to obstacles, several safety regions are observed by different sensor systems (Schäfer and Berns, 2006) (see Fig. 2). First of all, hindrances can be detected using the stereo camera system mounted at the front of the vehicle. The stereo camera's narrow field of vision is compensated by local obstacle memories to either side of the robot realising a short-term representation of detected obstacles. This obstacle detection facility is complemented with two laser range finders (field of vision: 180 degrees, angular resolution: 0.5 degrees, distance resolution: about 0.5 cm) monitoring the environment nearby the vehicle. Data from both sources of proximity data is used for obstacle avoidance by appropriate behaviours, the fusion of which is performed inside the behaviour network.In case of

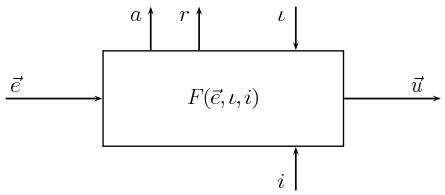

Figure 3: Basic behaviour module.

emergency, the system is stopped on collision by the safety bumpers which are directly connected to the emergency stop to ensure maximal safety.

3 BEHAVIOUR-BASED CONTROL SYSTEM OF RAVON

This section introduces the components used for building up the behaviour-based network controlling RAVON.

3.1 Behaviour Module

The fundamental unit of the proposed control architecture is the behaviour module (see Fig. 3). Each atomic behaviour is wrapped into such a module with a defined interface. Behaviours can be described as three-tuples of the form

$$\mathcal{B} = (r, a, F) \tag{1}$$

where r is the target rating function, a is the activity function, and F is the transfer function of the behaviour. Additionally each behaviour receives an input vector $\vec{e}$, an activation ι, and an inhibition i and generates an output vector $\vec{u}$.

More precisely behaviours receive data needed for fulfilling their work via the sensor input $\vec{e} \in \Re^n$ which can be composed of sensory data or information from other behaviours. The output vector $\vec{u} \in \Re^m$ transmits data generated by the behaviour. This output describes the influence a behaviour can have on the environment or on other behaviours.

Each behaviour owns an input determining its activation $\iota \in [0,1]$. In this notation $\iota = 0$ indicates deactivation and $\iota = 1$ a fully activated behaviour. Values between 0 and 1 refer to a partially activated behaviour. Activation can be used to adjust the relevance of competing behaviours. The inverse effect is achieved by inhibition $i \in [0,1]$ which is used to reduce the activation of a behaviour: $i = 1$ refers to full inhibition, $i = 0$ to no inhibition.

Information about the activity of a behaviour is provided by the output $a \in [0,1]$. The maximal activity is described by $a = 1$, inactivity by $a = 0$. It is defined by the activity function

$$a(\vec{e}, \iota, i) = a_{\text{int}}(\vec{e}) \cdot \iota \cdot (1 - i) \tag{2}$$

where $a_{\text{int}}(\vec{e}) \in [0,1]$ is an internal function representing the intended activity of the behaviour.

The target rating $r \in [0,1]$ deals as an indicator for the contentment of a behaviour. A value of $r = 0$ indicates that the behaviour is content with the actual state, while $r = 1$ shows maximal dissatisfaction.

The output vector $\vec{u}$ of a behaviour is determined using its transfer function $F(\vec{e}, \iota, i)$ where

$$F : \Re^n \times [0,1]^2 \to \Re^m, \quad F(\vec{e}, \iota, i) = \vec{u}$$

This function provides the intelligence of a behaviour, calculating actions depending on input values and internal representations. This can be a reactive respond to input values but also a more complex calculation as a state machine or sophisticated algorithms. Both reactive and deliberative behaviours can be implemented that way.

3.2 Example Behaviour `Roll Stop`

In this section a behaviour reacting on high roll is described in order to exemplify the behaviour properties described before. As the behaviour wants to stop the vehicle, the output $\vec{u}$ is a velocity of zero. Therefore the transfer function is:
$\vec{u} = v_{\text{out}} = 0$
This velocity has an effect if the activity a rises:
$a = \texttt{Threshold}(\texttt{roll}) \cdot \iota \cdot (1 - i)$
Here $\texttt{Threshold}(\texttt{roll})$ is a function returning 0 or 1 depending on the roll value being below or above a given threshold. The activity is scaled by the activation ι and the inhibition i as stated above. Similarly the target rating r is
$r = \texttt{Threshold}(\texttt{roll})$
In case of normal roll angles the behaviour is content ($r = 0$) while for high roll angles it is dissatisfied ($r = 1$).

3.3 Fusion Behaviour Module

In case of competing behaviours so called fusion behaviours (see figure 4) are used for coordination. The underlying assumption of the fusion of output values is that behaviours having a high activity deserve a higher influence on the control than those with lower activity. The interface of fusion behaviours implements a refinement of usual behaviours. For each of the competing behaviours $\mathcal{B}_i$ the activity (indicated by

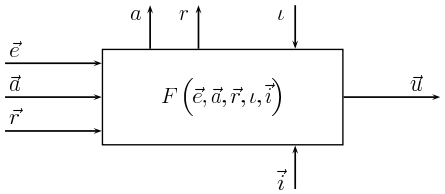

Figure 4: Fusion behaviour module.

$\vec{a}$), target rating (indicated by $\vec{r}$) and output vector $\vec{u}$ is provided. The output vector is fed into the fusion behaviour as $\vec{e}$. Additionally there is a fusion of inhibiting behaviours by the inhibition inputs $\vec{i}$. The transfer function then is the fusion function $f(\vec{a},\vec{e})$ which processes these input values to a merged output control vector $\vec{u}$.

The fusion function can have several implementations, in this work the weighted fusion is used: Here the control values are weighted with the activity of the corresponding behaviour, leading to a fusion function f_{weighted}, where

$$\vec{u} = f_{\text{weighted}}(a_0,\vec{u}_0,\ldots,a_{n-1},\vec{u}_{n-1}) = \frac{\sum_{j=0}^{n-1} a_j \cdot \vec{u}_j}{\sum_{k=0}^{n-1} a_k} \quad (3)$$

The activity is set according to the weighted input activities, the activation, and the maximally activated inhibiting behaviour:

$$a = \frac{\sum_{j=0}^{n-1} a_j^2}{\sum_{k=0}^{n-1} a_k} \cdot \iota \cdot (1 - i_m) \text{ where } i_m = \max_l(i_l)$$

The target rating of a fusion behaviour indicates its goal to satisfy highly activated input behaviours and is calculated as follows:

$$r = \frac{\sum_{j=0}^{n-1} a_j \cdot r_j}{\sum_{k=0}^{n-1} a_k}$$

The weighted fusion function provides a subtle gradation of coordinating behaviour control outputs regarding their activity.

3.4 Behaviour Network of RAVON

The behaviour network implemented on RAVON (see Fig. 5) comprises three control chains affecting desired rotation, sideward motion, and velocity of the vehicle. The rotational and the sideward components are influenced by obstacle avoidance behaviours. For safety reasons the velocity is adjusted according to obstacle proximity and critical vehicle inclination. In this behaviour network the following types of behaviours are used:

- Emergency Stop: Stop due to laser scanner data, positive or negative obstacles detected by the camera system; independently used for forward and backward motion.
- Slow Down: Reduce velocity due to obstacle proximity (laser scanner, positive/negative obstacles); independently used for forward and backward motion.
- Keep Distance Rotational: Turn away from obstacles (laser scanner, positive/negative obstacles); independently used for both sides of the vehicle.
- Keep Distance Sideward: Accomplish sideward motion due to obstacles at the side; independently used for both sides of the vehicle.
- Evasion: Evade obstacles at the front by arbitrating between the keep distance behaviours.
- Point Access: Accessing a given position.
- Point Access Ranking: Perform ranking manoeuvres accounting for kinematic constraints.
- Trace Back: Follow just driven path backwards in order to escape dead ends.

The advantage of this approach is the emergent vehicle behaviour leading to unforeseen, but suitable reaction on several external influences at a time. However, especially the maintenance of vehicle and person safety requires methods for guaranteeing fundamental characteristics of the vehicle motion e.g. in critical situations. Therefore, it is necessary to formally verify the behaviour network with respect to a given set of specifications.

4 THE AVEREST SYSTEM

In this section, we describe the Averest[1] framework (Schneider and Schuele, 2005; Schneider and Schuele, 2006) that provides tools for verifying temporal properties of synchronous programs (Benveniste et al., 2003; Halbwachs, 1993) as well as for compiling these programs to equivalent hardware and software systems. In particular, many formal verification techniques, including model checking of temporal properties of finite and infinite state systems

[1] http://www.averest.org

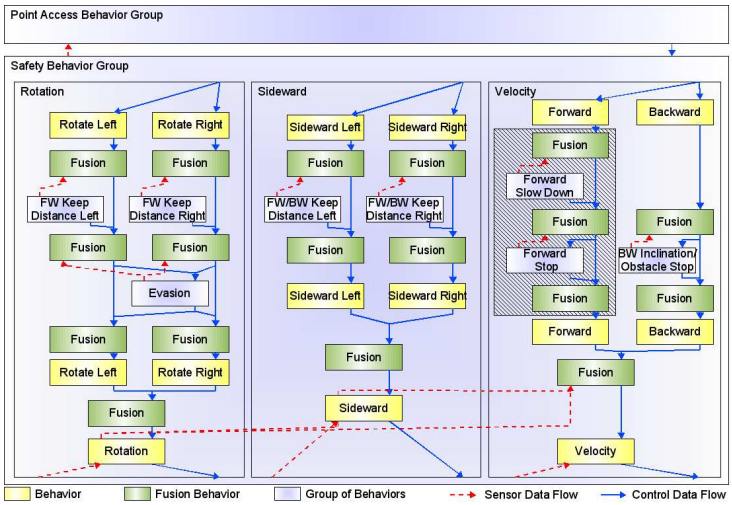

Figure 5: Behaviour network of RAVON.

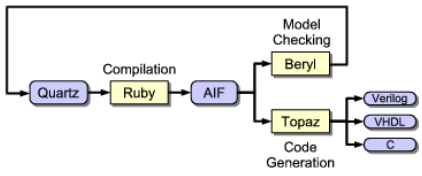

Figure 6: Averest design flow.

are available. In Averest, a system is described using the Esterel–like synchronous programming language Quartz (Schneider, 2001a), and specifications can be given in temporal logics such as LTL and CTL (Schneider, 2003). Currently, Averest consists of the following tools:

- ruby: a compiler for translating Quartz programs to finite and infinite state transition systems
- beryl: a symbolic model checker for finite and infinite state transition systems
- topaz: a code generator to convert transition systems into hardware and/or software

Figure 6 shows the typical design flow. A given Quartz program is first translated to a symbolically represented transition system in Averest's Interchange Format AIF that is based on XML. The AIF description can then be used for verification and code generation. Moreover, there are interfaces to third–party tools, e.g. other model checkers such as SMV (McMillan, 1992). In the remainder of this section, we describe some background information on Averest focusing on the verification of behaviour networks.

The basic paradigm of synchronous languages (Benveniste et al., 2003; Halbwachs, 1993) is the distinction between *micro and macro steps* in a program. From a programmer's point of view, micro steps do not take time, whereas macro steps take one unit of time. Hence, consumption of time is explicitly programmed by partitioning the program into macro steps. This programming model, referred to as perfect synchrony (Benveniste et al., 2003; Halbwachs, 1993), together with a deterministic form of concurrency allows the compilation of *multi–threaded synchronous programs* to deterministic single–threaded code. A distinct feature of synchronous languages is their *detailed formal semantics* that is usually given by means of transition rules in structural operational semantics. This makes synchronous languages attractive for safety–critical applications where formal verification is mandatory.

After translating a Quartz program to a transition system, it can be verified using symbolic model checking techniques. For that purpose, the specifications are given as a set of temporal logic formulas that describe the desired properties of the system. The most frequently used properties are safety and liveness properties. Intuitively, a safety property states that a condition invariantly holds on a given path of the transition system. Similarly, a liveness property states that a condition holds at least once on a given path. As an example for a safety property, the formula $\mathsf{AG}\varphi$ states that φ holds on all possible computation paths of the system. An example for a liveness property is the formula $\mathsf{EF}\varphi$, stating that there exists a path such that φ eventually holds.

A breakthrough in formal verification was achieved in the early nineties, where it was observed that finite sets can be efficiently represented by means of binary decision diagrams (BDDs), a canonical nor-

mal form for propositional logic formulas (Bryant, 1986). The development of BDDs was a cornerstone for *symbolic model checking* procedures based on fixpoint computations (Burch et al., 1990) (see textbooks like (Clarke et al., 1999; Schneider, 2003) for more details). With sophisticated implementations and refinements of symbolic model checking, it has become possible to verify systems of industrial size, and to detect errors that can hardly be found using simulation (Clarke and Wing, 1996).

5 FORMAL VERIFICATION OF THE BEHAVIOUR NETWORK

In case of RAVON, the most important behaviour network property is the control of the vehicle velocity due to obstacles or critical inclination. The behaviours affecting the control in this respect are marked hatched in Fig. 5. In order to formally verify this part of the behaviour network, the corresponding behaviours have been implemented in the synchronous language Quartz. In this way, the correctness of every single behaviour can be shown by means of a symbolic model checker. By forming a group of the mentioned behaviours, it is even possible to verify specifications concerning the overall behaviour of the complete velocity control part of the behaviour network. Moreover, the Quartz code can be exported to C code and can be wrapped into a module that replaces the original (unverified) code. As the output of the verified module is directly transferred to the vehicle motion actuators, it can be guaranteed that slowing down and stopping has the intended effect.

The verified and synthesized parts of RAVON's behaviour network are depicted in in Fig. 7. It shows the following structure: If none of the slow-down and stop behaviours is active, the velocity given by higher layers (*ci_velocity*) is piped through the three fusion behaviours without change to *co_velocity*. As soon as one of the mentioned behaviours becomes active (in case the inclination rises above a threshold or the obstacle distance becomes too low), the active behaviour uses its activity to inhibit the fusion behaviour which is above it. At the same time, it proposes a velocity of zero to the underlying fusion behaviour. This fusion behaviour calculates a weighted sum (using the input activities) of the input velocity values. The more active a behaviour becomes, the less active the fusion behaviour above is. Therefore, the influence of the slow-down behaviour rises and the velocity output of the underlying fusion behaviour decreases. This mechanism is implemented on two layers here.

For this behaviour network, we checked eight

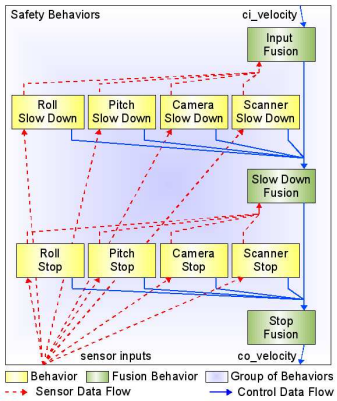

Figure 7: Structure of the verified part of RAVON's behaviour network.

specifications including the ones we list below. In this context, uppercase words indicate system parameters, `ci_` indicates controller inputs, `co_` controller outputs, and `si_` sensor inputs. Numbers are marked as unsigned integers (`0u`).

- The output velocity is never higher than the input velocity:

```
A G (co_velocity <= ci_velocity);
```

- In case of no near obstacle and tolerable inclination, the output velocity equals the input velocity. Therefore, the vehicle is not slowed down without a reason:

```
A G ((si_camera_distance
  >= MAX_VELOCITY_OBSTACLE_DISTANCE)
 & (si_scanner_distance
  >= MAX_VELOCITY_OBSTACLE_DISTANCE)
 & (si_roll
  <= MAX_VELOCITY_INCLINATION)
 & (si_pitch
  <= MAX_VELOCITY_INCLINATION)
 -> co_velocity == ci_velocity);
```

- If very high inclination or very near obstacles occur, the output velocity is set to zero, i.e., the vehicle stops:

```
A G (si_roll >= STOP_INCLINATION
 -> co_velocity == 0u);
A G (si_pitch >= STOP_INCLINATION
 -> co_velocity == 0u);
A G (si_camera_distance
  <= STOP_OBSTACLE_DISTANCE
 -> co_velocity == 0u);
A G (si_scanner_distance
  <= STOP_OBSTACLE_DISTANCE
 -> co_velocity == 0u);
```

- In case of a rising roll angle, the vehicle slows down. Similar specifications hold for the pitch angle and for near obstacles:

```
A G (ci_velocity == MAX_VALUE
```

```
& si_camera_distance
 > MIN_VELOCITY_OBSTACLE_DISTANCE
& si_scanner_distance
 > MIN_VELOCITY_OBSTACLE_DISTANCE
& si_pitch
 < MIN_VELOCITY_INCLINATION
& si_roll
 > MAX_VELOCITY_INCLINATION
& si_roll
 < MIN_VELOCITY_INCLINATION
-> co_velocity < MAX_VALUE
 & co_velocity > 0u );
```

In order to avoid vacuous specifications, formulas of the type $\mathsf{AG}(\varphi \rightarrow \psi)$ are always complemented with $\mathsf{EF}\varphi$ (not shown here). In this way, it is guaranteed that the antecedent of the implication is not always false. All of our specifications can be expressed in the temporal logic CTL. Hence, we can use state-of-the-art model checking techniques to verify these properties. Using global model checking, the model checkers perform a complete traversal of the reachable states and thereby check whether the given specifications hold. Using local model checking, the model checkers perform some sort of an induction proof to avoid a complete state space traversal.

BDD-based symbolic model checkers do usually not support floating point numbers. Therefore, integers with a given bitwidth are used for the implementation in Quartz instead of floating point numbers. In order to integrate the verified module in the control system, a conversion from floating point to fixpoint numbers is performed for control and sensor values.

In our case study, global model checking was able to check the specifications up to a sufficiently large bitwidth. We used CadenceSMV as backend of Averest. In the following table, we list experimental results for some bitwidths. For each bitwidth, we list the number of reachable states, the runtime that was necessary to verify the system, and the number of required BDD nodes for the entire verification of the eight specifications. All experiments were performed on a PentiumIV with 512 MByte main memory.

Table 1: Experimental results of the verification process.

bits	states	runtime (s)	BDD nodes
3	1024	0.95	453
4	8192	1.59	10012
5	65536	2.54	10063
6	524288	3.87	10521
7	4194204	5.54	15399
8	33554432	8.85	49500

In the process of implementing the behaviour network, the question arose if the fusion behaviours could implement a maximum fusion function (i.e. the most active behaviour has full influence) instead of the weighted fusion function. By means of formal verification, it was possible to show that in this case not all specifications were valid. Depending on the implementation, there was either no slowing down of the vehicle (but only abrupt stopping) or it was possible that the velocity output was higher than the velocity input. Experimental changes concerning the structure and implementation of the behaviour network can therefore be performed with immediate feedback about the correct properties stated in the specifications.

6 CONCLUSIONS AND FUTURE WORK

We presented an approach to the formal verification of a behaviour-based control network. Without the need of testing, it is guaranteed that the specified properties are valid for all possible input traces. Of course, it is necessary to verify more parts of the system or even the complete behaviour-based network. Due to the enormous number of states, this inevitably leads to the need of improving the model checking approach. Therefore, methods like modular model checking and abstraction will have to be analysed in this respect. The uniformity of the behaviours is seen to be an advantage in this context that can be exploited by tailored verification and abstraction techniques.

ACKNOWLEDGEMENTS

The research work presented in this paper is funded by the German Federal State of Rhineland-Palatinate within the excellence cluster ”Dependable Adaptive Systems and Mathematical Modeling”.

REFERENCES

Benveniste, A., Caspi, P., Edwards, S., Halbwachs, N., Le Guernic, P., and de Simone, R. (2003). The synchronous languages twelve years later. *Proceedings of the IEEE*, 91(1):64–83.

Berry, G. (1998). The foundations of Esterel. In Plotkin, G., Stirling, C., and Tofte, M., editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT.

Bryant, R. (1986). Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691.

Burch, J., Clarke, E., McMillan, K., Dill, D., and Hwang, L. (1990). Symbolic model checking: 10^{20} states and beyond. In *Symposium on Logic in Computer Science (LICS)*, pages 1–33, Washington, D.C. IEEE Computer Society.

Clarke, E., Grumberg, O., and Peled, D. (1999). *Model Checking*. MIT, London, England.

Clarke, E. and Wing, J. (1996). Formal methods: State of the art and future directions. Technical Report CMU-CS-96-178, Carnegie Mellon University. ftp://reports.adm.cs.cmu.edu/usr/anon/1996/CMU-CS-96-178.ps.

Diethers, K., Firley, T., Krger, T., and Thomas, U. (2003). A new framework for task oriented sensor based robot programming and verification. In *International Conference on Advanced Robotics (ICAR)*, pages 1208–1214, Coimbra,Portugal. IEEE Computer Society.

Halbwachs, N. (1993). *Synchronous programming of reactive systems*. Kluwer.

Kim, M. and Kang, K. (2005). Formal construction and verification of home service robots: A case study. In Peled, D. and Tsay, Y.-K., editors, *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, volume 3707 of *LNCS*, pages 429–443, Taipei, Taiwan. Springer.

McMillan, K. (1992). The SMV system, symbolic model checking - an approach. Technical Report CMU-CS-92-131, Carnegie Mellon University.

Schäfer, H. and Berns, K. (2006). Ravon - an autonomous vehicle for risky intervention and surveillance. In *International Workshop on Robotics for risky intervention and environmental surveillance - RISE*.

Schmitz, N., Proetzsch, M., and Berns, K. (2006). Pose estimation in rough terrain for the outdoor vehicle ravon. In *37th International Symposium on Robotics (ISR)*.

Schneider, K. (2001a). Embedding imperative synchronous languages in interactive theorem provers. In *Conference on Application of Concurrency to System Design (ACSD)*, pages 143–156, Newcastle upon Tyne, UK. IEEE Computer Society.

Schneider, K. (2001b). *Exploiting Hierarchies in Temporal Logics, Finite Automata, Arithmetics, and μ-Calculus for Efficiently Verifying Reactive Systems*. Habilitation Thesis. University of Karlsruhe.

Schneider, K. (2003). *Verification of Reactive Systems – Formal Methods and Algorithms*. Texts in Theoretical Computer Science (EATCS Series). Springer.

Schneider, K. (2006). The synchronous programming language Quartz. Internal Report (to appear), Department of Computer Science, University of Kaiserslautern.

Schneider, K. and Schuele, T. (2005). Averest: Specification, verification, and implementation of reactive systems. In *Conference on Application of Concurrency to System Design (ACSD)*, St. Malo, France. participant's proceedings.

Schneider, K. and Schuele, T. (2006). A framework for verifying and implementing embedded systems. In Straube, B. and Freibothe, M., editors, *Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 242–247, Dresden, Germany. GI/ITG/GMM, Fraunhofer Institut fr Integrierte Schaltungen, ISBN 3-9810287-1-6.

Schuele, T. and Schneider, K. (2006). Bounded model checking for infinite state systems. *Formal Methods in System Design (FMSD)*. DOI 10.1007/s10703-006-0019-9.

Sharygina, N., Browne, J., Xie, F., Kurshan, R., and Levin, V. (2004). Lessons learned from model checking a NASA robot controller. *Formal Methods in System Design (FMSD)*, 25(2-3):241–270.

Sowmya, A., So, D., and Tang, W. (2002). Design of a mobile robot controller using Esterel tools. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 65(5). Workshop on Synchronous Languages, Applications, and Programming (SLAP).