# Quantitative Aspects of Behaviour Network Verification

Christopher Armbrust, Thorsten Ropertz, Lisa Kiekbusch, and Karsten Berns

Robotics Research Lab, Department of Computer Science,
University of Kaiserslautern, P.O. Box 3049, 67653 Kaiserslautern, Germany
{armbrust,ropertz,kiekbusch,berns}@cs.uni-kl.de
http://rrlab.cs.uni-kl.de/

**Abstract.** This paper presents quantitative aspects of an approach for the modelling and verification of behaviour networks published previously and describes the application of said modelling technique to a complex coordinating behaviour. In order to decrease the number of interconnection failures in behaviour networks, verification techniques focusing on behaviour interaction can be applied. In previous work, the authors have introduced a novel approach for modelling behaviour networks as networks of finite-state automata, to which model checking can be applied as verification technique. This paper presents how the approach can be used to model complex behaviours and provides calculations of the numbers of states, transitions, and state variables in the resulting automata.

**Keywords:** Behaviour-based System; Behaviour Network; Behaviour Modelling; Behaviour Network Verification; Quantitative Aspects

## 1 Introduction

Behaviour-based robot control systems (BBS) have shown to provide several advantages: In contrast to traditional approaches, complex tasks are decomposed into simple sub-goals pursued by individual behaviours. Due to the behaviours' limited complexity, they are easier to develop, implement, and maintain. The restricted scope also facilitates the reusability, such that common functionalities can be realised based on well-tested modules.

Behaviours interact via signal exchange in order to provide a suitable overall behaviour. With a growing demand for numerous functionalities, the behaviour network complexity increases and the necessity of sophisticated analysis methods and tools to ensure a certain level of quality arises. Especially regarding safety critical systems, experience has shown that proving the systems' correctness is obligatory. A common method for proving correctness is formal verification, which has to be adapted to the properties of BBS in order to become feasible.

This paper continues the research on behaviour network verification conducted at the Robotics Research Lab. In [1] and [2], a novel approach for modelling behaviour-based networks as networks of finite-state automata has been

presented. The purpose of this paper is to provide information about how to model more complex behaviours and about the complexity of the created models. This is done by explaining how a special coordinating behaviour can be properly modelled using the presented approach and by providing information about the number of locations, edges, and variables in the created models.

## 2 Related Work

Model checking is a common technique for ensuring the correctness of concurrent finite-state systems. Thereby, an abstract model of the system is created and its compliance with a given specification is checked by applying a verification algorithm (see [4] for an overview). The authors of [13] describe the use of model checking for verifying the control system of an unmanned aircraft. In [8] the application and extension of model checking techniques for verifying spacecraft control software is presented. Another example of the use of model checking is given in [7], which applies model checking to verify a distributed coordination algorithm in the field of swarm robotics.

In general, different types of models are possible depending on the model checking method. In [5], X-machines (a computational machine resembling an FSM extended by memory) are proposed for modelling an agent's behaviour. Thereby, each agent is represented by a single X-machine, which leads to large monolithic elements and thus complicates the modelling. The authors of [10] used the synchronous programming language Quartz, which allows for an automatic derivation of a fine-grained model, for implementing single behaviours and to prove their correctness using model checking. UPPAAL is a model checking tool that requires the system to be modelled by automata. In [12] model checking with UPPAAL in combination with fault-tree analysis is applied to prove the absence of critical events in the system. The correctness and completeness of fault-trees can also be verified using model checking as described in [6].

## 3 Behaviour Network Modelling and Verification

This section introduces the behaviour-based architecture used for the work at hand (the iB2C) and describes how iB2C networks can be modelled and verified.

The presented work uses the behaviour-based architecture iB2C, which has been implemented using the software frameworks MCA2-KL[1] and FINROC[2]. The iB2C is described extensively in [9], while only its essential aspects are presented here. An iB2C behaviour is defined as $B = (f_a, f_r, F)$, with $f_a$ calculating its *activity vector* $\boldsymbol{a}$, $f_r$ calculating its *target rating* $r$, and $F : \boldsymbol{u} = F(\boldsymbol{e}, \iota)$ *transferring* its input vector $\boldsymbol{e}$ and activation $\iota$ into the output vector $\boldsymbol{u}$. The activation is a combination of a behaviour's *stimulating* ($s$) and *inhibiting* $i = \|\boldsymbol{i}\|_\infty$ inputs and is calculated as $\iota = s \cdot (1 - i)$. A behaviour indicates the amount of influence

[1] MCA2-KL: Modular Controller Architecture Version 2 - Kaiserslautern Branch
[2] FINROC is the downward compatible successor of MCA2-KL (see [11]).

it intends to have in a network using its *activity* $a$. According to an iB2C principle, a behaviour's activation limits its activity, i.e. $a \leq \iota = s \cdot (1 - i)$. The target rating $r$ describes how satisfied a behaviour is with the current situation. $s$, $i$, $a$, and $r$ are limited to $[0, 1]$. The typical connection between two behaviours is that one behaviour stimulates or inhibits another with its activity. Furthermore, fusion behaviours (FB) can be used to coordinate competing behaviours. Figure 6 depicts two iB2C networks.

iB2C networks are modelled as networks of finite-state automata, on which model checking is performed using the UPPAAL[3] toolbox. In UPPAAL, a network of *automata* (parametrised instantiations of *templates*) is called *system*. An automaton consists of *locations* and interconnecting *edges*. Whether an edge can be taken is restricted by *guards* (side-effect free Boolean expressions) and synchronisations. The latter are realised via so-called *channels*. Edges can be labelled with a channel name followed by "!" for a sending or "?" for a receiving channel. Furthermore, so-called *updates* (assignments) can be added to edges. Figure 3 shows an automaton with the described elements. When modelling an iB2C network, each behaviour is represented by an instantiation of each of five basic templates, namely `StimulationInterface`, `InhibitionInterface`, `ActivationCalculation`, `ActivityCalculation`, and `TargetRatingCalculation`. Three of the five basic templates are shown in Figs. 1 to 3. Due to reasons of complexity, the value range of the behaviours is limited to the set of $\{0, 1\}$. Experience has shown that this does not pose a problem to the effectiveness of the presented approach.

As `InhibitionInterface` depends on the number of inhibiting behaviours, different versions for different numbers of inhibiting behaviours are created by the modelling algorithm, while the other basic templates are just instantiated and connected using the appropriate channels. The structure of the different versions of `InhibitionInterface` is a hypercube that has the number of inhibiting behaviours as dimension. To verify the proper operation of a BBS, queries are given to UPPAAL's model checker, which evaluates them to true or false.

## 4 Modelling Complex Behaviour Nodes

As an example of how to model the activity function of a complex behaviour node in a more precise way than depicted in Fig. 2, the modelling of a maximum fusion behaviour shall be described here. For an FB $B_{\text{Fusion}}$ with $n_c$ connected behaviours $B_{\text{Input}_d}$, $a_{\text{Fusion}} = 1$ if $\left(\exists d : (1 \leq d \leq n_c) \wedge \left(a_{\text{Input}_d} = 1\right)\right)$ and $(\iota_{\text{Fusion}} = 1)$, otherwise $a_{\text{Fusion}} = 0$.

A naive implementation of a template for calculating $f_a$ of an FB would result in a basic structure resembling that of a hypercube with dimension $n_c + 1$—one for each connected input behaviour and one for monitoring $\iota_{\text{Fusion}}$ (Version 1). In order to reduce the number of locations and transitions (see Sec. 5), the modelling of $f_a$ of an FB has been split up into different automata: For each connected behaviour $B_{\text{Input}_d}$, an instance of `FBIBActivityChanged` (see Fig. 4) is created.

[3] see `http://www.uppaal.org/` and [3]

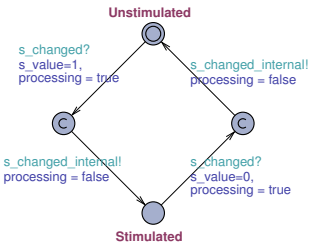



**Fig. 1.** `StimulationInterface` of a single behaviour.

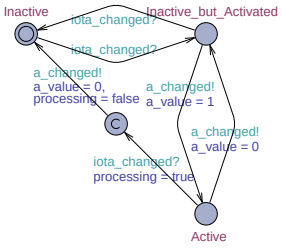



**Fig. 2.** `ActivityCalculation` (calculating the activity).

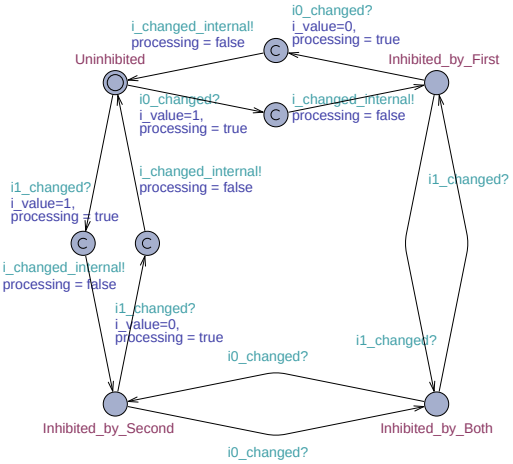



**Fig. 3.** `InhibitionInterface` for a behaviour that is inhibited by two others (double circle: initial location; "C" in circle: committed location; purple: name of location; turquoise: channel; blue: update).

Furthermore, there is one single instance of `FBActivityCalculation` (Version 2) that combines the signals of the individual automata. Each instance of `FBIBActivityChanged` monitors $\iota_{\text{Fusion}} = 1$ and the activity of the corresponding input behaviour. If both equal 1, it transitions to location `Active` and sends a signal to the combining automaton. In contrast to `FBIBActivityChanged`, `FBActivityCalculation` (Version 2) depends on $n_c$ as it has to process signals from $n_c$ different automata. Figure 5 depicts this automaton for the case of $n_c = 2$. In the initial location, it is assumed that $B_{\text{Fusion}}$ is inactive, i.e. $a_{\text{Fusion}} = 0$. If one of the instances of `FBIBActivityChanged` signals that the corresponding input behaviour is active *and* $\iota_{\text{Fusion}} = 1$, then `fb_a_value` is set to 1 and this change is signalled via `fb_a_changed`. If now the second instance of `FBIBActivityChanged` also indicates that its corresponding input behaviour is active, another change of location takes place—but no change of `fb_a_value` and no signalling is performed. In case of an instance signalling a change now, `FBActivityCalculation` does not signal a change or update a variable. But if then the other instance of `FBIBActivityChanged` also signals a change, `FBActivityCalculation` goes back to its initial location, setting `fb_a_value` to 0 again and signalling the change via `fb_a_changed`. The structure of `FBActivityCalculation` for $n_c$ connected input behaviours is a hypercube with dimension $n_c$. Just like the different versions of `InhibitionInterface`, it is automatically created by the modelling algorithm.

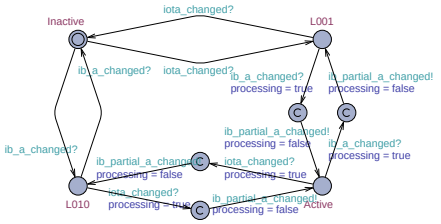



**Fig. 4.** `FBIBActivityChanged` of a fusion behaviour.

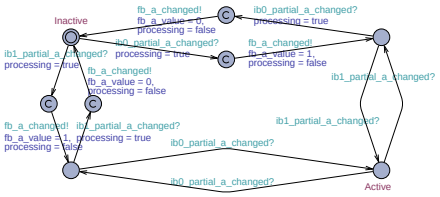



**Fig. 5.** `FBActivityCalculation` (Version 2) for a fusion behaviour with two connected input behaviours.

## 5 Quantitative Aspects

The number of locations and edges in a model for a given behaviour network provides an estimate of the complexity of the model. For the basic templates, the locations and edges can simply be counted (see Tab. 1). As already mentioned in Sec. 3, `InhibitionInterface` for $n_i$ inhibiting behaviours has the basic structure of a hypercube of dimension $n_i$. The number of vertices in such a hypercube is $2^{n_i}$. It has $2^{n_i-1} \cdot n_i$ edges. As the edges in `InhibitionInterface` are bidirectional, this has to be multiplied by 2, resulting in $2^{n_i} \cdot n_i$. For each dimension (i.e. for each inhibiting behaviour), there are two additional committed locations with two additional edges. Summing that up yields $2^{n_i} + 2 \cdot n_i$ locations and $n_i \cdot (2^{n_i} + 2)$ edges. The same numbers can be calculated for `FBActivityCalculation` (Version 2) and `FBTargetRatingCalculation`. As mentioned in Sec. 4, the basic structure of `FBActivityCalculation` (Version 1) is a hypercube with dimension $n_c + 1$. Additional committed locations are added for the following cases: (1) $\iota_{\text{Fusion}}$ changes from 0 to 1 or vice versa *and* at least one of the competing behaviours is active ($2 \cdot (2^{n_c} - 1)$ cases). (2) The activity of one competing behaviour changes from 0 to 1 or vice versa *and* $\iota_{\text{Fusion}} = 1$ and *none* of the other competing behaviours is active ($2 \cdot n_c$ cases). This results in a total of $2^{(n_c+2)} + 2 \cdot n_c - 2$ locations. Taking into account that the edges of `FBActivityCalculation` (Version 1) are bidirectional and for every additional committed location one edge is added yields as total number of edges $2^{(n_c+1)} \cdot (n_c + 2) + 2 \cdot (n_c - 1)$. For $n_c \leq 3$, `FBActivityCalculation` (Version 1) needs less locations and for $n_c \leq 2$, it also needs less edges than `FBActivityCalculation` (Version 2). However, in the current implementation, Version 2 is always used.

Additionally, five variables of type `bool` (`processing` flags) and four of type `int[0,1]` (for values of behaviour signals) are needed for each behaviour along with different channels. There are three binary channels (for internal signalling of changes of $s$, $i$, and $\iota$) and two broadcast channels (for signalling changes of $a$ and $r$). No channels are needed for signalling changes of $s$ or $i$ from other behaviours as these are the channels signalling a change of $a$ of the respective stimulating or inhibiting behaviour. For an FB with $n_c$ competing behaviours,

**Table 1.** The number of locations and edges of each template ($n_i$: number of inhibiting behaviours; $n_c$: number of competing behaviours).

| Template | #Locations | #Edges |
|---|---|---|
| `StimulationInterface` | 4 | 4 |
| `InhibitionInterface` | $2^{n_i} + 2 \cdot n_i$ | $n_i \cdot (2^{n_i} + 2)$ |
| `ActivationCalculation` | 4 | 7 |
| `ActivityCalculation` | 4 | 6 |
| `TargetRatingCalculation` | 2 | 2 |
| `FBActivityCalculation` (V. 1) | $2^{(n_c+2)} + 2 \cdot n_c - 2$ | $2^{n_c+1} \cdot (n_c + 2) + 2 \cdot (n_c - 1)$ |
| `FBIBActivityChanged` | 8 | 12 |
| `FBActivityCalculation` (V. 2) | $2^{n_c} + 2 \cdot n_c$ | $n_c \cdot (2^{n_c} + 2)$ |
| `FBTargetRatingCalculation` | $2^{n_c} + 2 \cdot n_c$ | $n_c \cdot (2^{n_c} + 2)$ |

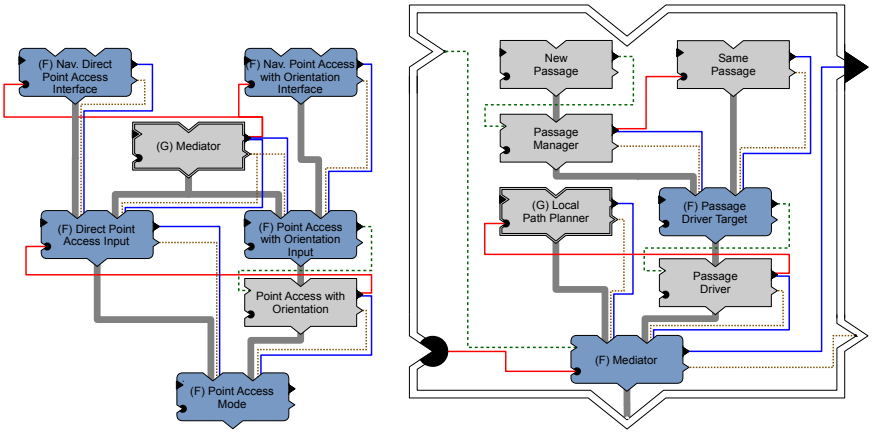
$n_c$ binary channels for signalling between an instance of `FBIBActivityChanged` and `FBActivityCalculation` as well as $n_c$ `processing` flags for the $n_c$ instances of `FBIBActivityChanged` have to be added.

## 6 Application Example

In this section, the modelling and verification of the behaviour networks presented in [1] (see Figs.6a and 6b) shall be investigated further using the results of Secs. 4 and 5. *(G) Drive Control* with the sub-network *(G) Mediator* shall control the motion of an autonomous off-road vehicle based on different navigation approaches. Due to page restrictions, the reader has to be referred to [1] for details. What is relevant here is that the BBS shall fulfil four requirements. That this is the case was proven using model checking.

*(G) Drive Control* (excluding the contents of *(G) Mediator*) consists of a standard behaviour, three FBs with two input behaviours, and two FBs with no input behaviours that are taken into account for this analysis. Three of the FBs are inhibited by another behaviour. In total, this results in 166 locations, 230 edges, 40 variables of type `bool`, 24 of type `int[0,1]`, 28 binary channels, and 12 broadcast channels. *(G) Mediator* alone consists of five normal behaviours (*(G) Local Path Planner* is considered as normal behaviour here.) and two FBs with two input behaviours each. Two of the normal behaviours are inhibited by another behaviour. This yields 158 locations, 221 edges, 39 variables of type `bool`, 28 of type `int[0,1]`, 25 binary channels, and 14 broadcast channels. Due to a number of optimisations in the modelling algorithm, the actual numbers are partly slightly lower. However, these numbers show that even simple behaviour networks can result in rather large networks of automata.

Several queries have been verified with UPPAAL to check the fulfilment of the aforementioned requirements. For each requirement, `verifyta` of the UPPAAL suite was called with its default values and passed the model of the system and the query file corresponding to the requirement. Table 2 shows CPU and memory usage on an Intel® Core™ i7 920 CPU @ 2.67 GHz with 12 GB RAM. The figures show that even the solving of rather simple queries leads to a considerable




(a) *(G) Drive Control* (b) *(G) Mediator*

**Fig. 6.** The two networks (grey node: simple behaviour; blue: maximum fusion behaviour; double-bordered grey: behavioural group; dashed green edge: stimulation; red: inhibition; blue: activity transfer; dotted brown: target rating; bold grey: data; filled stimulation input: always stimulated, i.e. $s = 1$).

**Table 2.** CPU and memory usage for verifying the requirements.

| Req. | #Queries | CPU User Time (Sum) | Virtual Memory (Max) |
|---|---|---|---|
| 1 | 5 | 5.950 ms | 86.944 KiB |
| 2 | 3 | 5.880 ms | 85.624 KiB |
| 3 | 1 | 1.890 ms | 84.300 KiB |
| 4 | 2 | 2.170 ms | 84.964 KiB |
| Sum | 11 | 15.890 ms | - |

memory consumption. Further experiments with larger networks (see [2]) have indicated that this is currently the main drawback of the presented approach and will have to be dealt with in the context of future work.

## 7 Conclusion and Future Work

This paper has demonstrated how a novel approach for modelling behaviour networks with the aim of verification can be applied to complex behaviours. Quantitative aspects of said approach have been presented and illustrated using the verification of a behaviour network from the control system of a real off-road vehicle. Future work will deal with optimising the modelling algorithm in order to reduce CPU and memory requirements of the verification process. Furthermore, the approach shall be used to verify more complex networks, taking into account aspects of the robot's environment.

**Acknowledgements** The authors gratefully acknowledge Prof. Roland Meyer from the Concurrency Theory Group[4] of the University of Kaiserslautern for his helpful comments and suggestions. Furthermore, the authors acknowledge the technical work done by the student Maryla Rittmann. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 285417.

## References

1. Armbrust, C., Kiekbusch, L., Ropertz, T., Berns, K.: Verification of behaviour networks using finite-state automata. In: Glimm, B., Krüger, A. (eds.) KI 2012: Advances in Artificial Intelligence. Springer, Saarbrücken, Germany (September 24-27 2012)
2. Armbrust, C., Kiekbusch, L., Ropertz, T., Berns, K.: Tool-assisted verification of behaviour networks. In: ICRA 2013. Karlsruhe, Germany (May 6-10 2013)
3. Behrmann, G., David, A., Larsen, K.G.: A tutorial on Uppaal. In: Bernardo, M., Corradini, F. (eds.) Formal Methods for the Design of Real-Time Systems. LNCS, vol. 3185, pp. 200–236. Springer (2004)
4. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
5. Eleftherakis, G., Kefalas, P., Sotiriadou, A., Kehris, E.: Modeling biology inspired reactive agents using x-machines. In: Okatan, A. (ed.) International Conference on Computational Intelligence 2004 (ICCI '04). pp. 93–96. International Computational Intelligence Society, Istanbul, Turkey (December 17–19 2004)
6. Faber, J.: Fault tree analysis with Moby/FT. Tech. rep., Department for Computing Science, University of Oldenburg (2005), publication available at `http://csd.informatik.uni-oldenburg.de/~jfaber/dl/ToolPresentationMobyFT.pdf`
7. Juurik, S., Vain, J.: Model checking of emergent behaviour properties of robot swarms. Proceedings of the Estonian Academy of Sciences 60(1), 48–54 (2011)
8. Lowry, M., Havelund, K., Penix, J.: Verification and validation of AI systems that control deep-space spacecraft. In: Ras, Z., Skowron, A. (eds.) Foundations of Intelligent Systems. LNCS, vol. 1325, pp. 35–47. Springer Berlin / Heidelberg (1997)
9. Proetzsch, M.: Development Process for Complex Behavior-Based Robot Control Systems. RRLab Dissertations, Verlag Dr. Hut (2010)
10. Proetzsch, M., Berns, K., Schuele, T., Schneider, K.: Formal verification of safety behaviours of the outdoor robot ravon. In: Zaytoon, J., Ferrier, J.L., Andrade-Cetto, J., Filipe, J. (eds.) ICINCO 2007. pp. 157–164. INSTICC Press (May 2007)
11. Reichardt, M., Föhst, T., Berns, K.: On software quality-motivated design of a real-time framework for complex robot control systems. In: Proceedings of the 7th International Workshop on Software Quality and Maintainability (SQM), in conjunction with the 17th European Conference on Software Maintenance and Reengineering (CSMR) (March 5 2013)
12. Schäfer, A.: Combining real-time model-checking and fault tree analysis. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003: Formal Methods. LNCS, vol. 2805, pp. 522–541. Springer Berlin / Heidelberg (2003)
13. Webster, M., Fisher, M., Cameron, N., Jump, M.: Model checking and the certification of autonomous unmanned aircraft systems. Tech. Rep. ULCS-11-001, University of Liverpool Department of Computer Science (2011)

[4] `http://concurrency.cs.uni-kl.de/`