# A Scalable Incomplete Test for the Boundedness of UML RT Models

Stefan Leue and Richard Mayr and Wei Wei

Department of Computer Science
Albert-Ludwigs-University Freiburg
Georges-Koehler-Allee 51, D-79110 Freiburg, Germany
E-mail: [leue|mayrri|wwei]@informatik.uni-freiburg.de

**Abstract.** We describe a scalable incomplete boundedness test for the communication buffers in UML RT models. UML RT is a variant of the UML modeling language, tailored to describing asynchronous concurrent embedded systems. We reduce UML RT models to systems of communicating finite state machines (CFSMs). We propose a series of further abstractions that leaves us with a system of linear inequalities. Those represent the message sending and receiving effect that the control flow cycles of every process have on the overall message buffer. The test tries to establish the existence of a linear combination of the effect vectors so that at least one message can occur an unbounded number of times. We discuss the complexity of this test and present experimental results using the IBOC system that we are implementing. Scalability of the test is in part due to the fact that it is polynomial for the type of sparse control flow graphs that are derived from UML RT models. Also, the analysis is local, i.e., it avoids the combinatorial state space explosion due to concurrency of the models. We also present a method to derive upper bound estimates for the maximal occupancy of each individual message buffer. While we focus on the analysis of UML RT models, the analysis can directly be applied to any type of CFSM models.

## 1 Introduction

The unboundedness of the communication channels in a communicating finite state machine (CFSM) model can have several negative effects. First, if the model represents a software design, the unboundedness of one or more of the communication channels hints at a possible design fault. For instance, the overflow of a communication buffer can have equally negative ramifications on the sanity of a system as, say, an overflow of the program heap due to inadequate object allocation and deallocation. Of course, unboundedness of a buffer can also be due to the environment, e.g., if it is flooded with requests from the outside. In this case, it is important to determine whether the unboundedness of certain buffers is only due to external flooding or to internal design flaws. Finally, buffers with unbounded capacity impede automated finite state analyzability since they induce an infinite state space that renders state space exploration incomplete in finite time.

In spite of the potential unboundedness of the buffers in CFSM systems one commonly observes that for many actual CFSM models the buffer occupancy is bounded by some small constant $k$. This is not surprising since, as we argue above, the unbounded growth of the buffers that are typically employed to implement communication channels is an undesired property of the system. If $k$-boundedness is proven, then one can safely replace the unbounded message buffers by $k$-bounded message buffers without changing the behavior of the system. Ideally, one wants to find individual bounds $k_i$ for every buffer $B_i$. A system with $k$-bounded buffers is a finite-state system, modulo any remaining infinity due to data.

Practitioners usually notice the $k$-boundedness of a system either by manual inspection of the code (for small examples), or by running random simulations. Both these methods are not reliable and do no scale. The objective of our paper is to present algorithms that are capable of establishing the boundedness of CFSM models in an automated fashion. In their seminal paper [8], Brand and Zafiropulo showed that for CFSM systems with unbounded buffers many interesting properties, including reachability and boundedness, are undecidable. Consequently, the boundedness analysis that we propose in this paper is inevitably an incomplete test. We use an over-approximation of CFSMs for which boundedness is decidable and for which bounds on the buffer length can be computed. By the very nature of over-approximations, not every bounded CFSM can be detected as such by this method and the obtained bounds are not necessarily optimal. However, the computed bounds are certainly upper bounds, which is sufficient to make the system analyzable by finite-state verification methods.

While our results apply to the whole class of CFSM systems, in this paper we are interested in a particular instance of this paradigm. Variants of CFSMs form the foundation of many object-oriented modeling techniques for concurrent, reactive systems. We will focus on the modeling notation UML RT [31]. UML RT is an integral part of the current version 2.0 of the Unified Modeling Language (UML) [29]. UML RT enjoys widespread use in the design of asynchronous distributed embedded systems [18, 28], and as an architectural description language [4]. Our interest in boundedness analysis for UML RT models is partly due to the availability of a commercial CASE tool supporting this modeling language. The *Rational Rose RealTime* tool suite, a direct successor to the ObjecTime Developer toolset [23], permits the graphical editing, interactive simulation and automated target platform implementation of UML RT models.

As we alluded to above, obtaining a boundedness result for a given UML RT model provides at least two benefits. First, it is an assurance of the well-formedness of the inter-object communication mechanism in the model. Second, the boundedness property of an UML RT model facilitates the translation of the model to a finite state verification tool such as the model checker SPIN [19]. SPIN requires all communication channels to have finite, compile-time known capacity limits. Having to commit to a specific channel capacity at modeling time may be undesirable. On the other hand, the fact that the boundedness of the UML RT model has been proven means that the designer can completely verify properties for models that are bounded by a sufficiently large buffer capacity limit. As we will explain later, the estimation of an actual upper bound is more intricate to obtain than the boundedness result, but we will present overapproximations that conservatively estimate buffer capacities for bounded models.

## 2 UML RT

UML RT has its root in the graphical modeling language ROOM [30]. ROOM has later been reconciled with the UML standard to form a UML compatible language for the modeling of real-time systems [31]. We will refer to this notation as *UML RT*. UML RT permits the description of the communication structure and the dynamic behavior of the systems. A system is decomposed into a set of concurrent active objects, called *capsules*. Capsules can be decomposed hierarchically into sub-capsules. The communication interfaces of the capsules are called *ports*. Ports can be associated with each other using *connectors* - the presence of a connector between ports indicates a communication channel. Inter-capsule communication is by message passing only, i.e., no shared variable communication is defined in UML RT.

The behavior of each capsule is described using a communicating, extended, hierarchical finite state machines (CEHFSM). These state machines are derived from ROOM-Charts [30] which, in turn, are a variant of Statecharts [17]. However, as opposed to Statecharts, the CEHFSMs in UML RT are strictly sequential, i.e., the orthogonality concept of Statecharts in absent in UML RT. The operational semantics of UML RT is characterized by two salient features: state machine transitions can only be triggered by message reception, and arbitrary programming language code can be executed in the course of a transition between states. The structure of this transition code is not specifically constrained by the UML RT definition, and, in fact, the Rose RealTime tool allows arbitrary C++ or Java code to be attached to transitions.

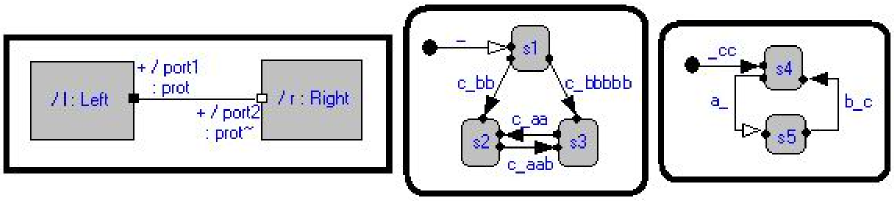


**Fig. 1.** The *2-Capsule* UML RT Model

Consider the simple UML RT model with two capsules given in Figure 1. It consists of two capsules, named `Left` and `Right`. Capsules represent active objects that may have state machines attached to them. Ports denote communication interfaces of capsules, and a connector between ports, such as the one between `port1` and `port2`, represents a bi-directional communication channel. Figure 1 also illustrates the state machines associated with the two capsules. Since in UML RT the transition labels are mere names that carry no semantics, we have used speaking transition names which indicate which message sending and receiving primitives are executed in the course of the respective transition. For instance, the transition labeled with $c\_aab$ consumes a message $c$. It then sends two messages $a$ and then a message $b$. For a more complete description of the UML RT notation we refer the reader to [31].

At the time of writing, there is no complete formal operational semantics for UML RT available in the literature that we are aware of. It turns out that the precise definition of a formal semantics is not a prerequisite for the work pursued here. We will present an approach that is taking advantage of a significant amount of abstraction in preparation of the analysis. These abstractions amount to an over-approximation of the actual system behavior so that subtle issues in the UML RT semantics, such as the ordering of message events and the treatment of message priorities, are not meaningful in the abstract system and hence for our analysis. In fact, our abstraction and the results of our algorithm are safe w.r.t. effects like message reordering and message loss in the channels. The only really necessary assumption is that no *unbounded* message duplication occurs. (Our boundedness test (YES/NO) is safe w.r.t. *finite* message duplication, but the computed upper bounds on the buffer lengths are not.)

## 3 Overview of the Boundedness Test

The underlying idea of our boundedness test is to determine whether at all it is possible to combine the cyclic executions of all of the processes in a UML RT model in such a way that the filling of at least one of the message buffers can be "blown up" in an unbounded way. Note that any infinite execution of the system can be understood as the combination of an infinite number of control state cycles through the CFSMs.

Consider the examples in Figure 2. All examples consist of two state machines which we assume to represent concurrent CFSMs. The state transition labels indicate message sending and receiving in the usual format. Since we are only interested in infinite execution sequences, all finite prefixes, e.g., transitions initializing the system, have been disregarded. In Example 1 it is easy to see that the system is unbounded. Any execution of the cycle through state `S1` will consume a message of type `a` and produce two messages, `b` and `c`. However, each one of these messages only produces another message of type `a` when cycling through `S2`. To the contrary, Example 2 represents a bounded system since an `a` message generates a single `b` or `c` message, while the consumption of a `b` or a `c` message triggers the generation of a single `a` message. Example 3 contains a spontaneous transition that generates a message of type `c` which may obviously flood one of the system's buffers in an unbounded fashion. Assessing the boundedness of Example 4 is less obvious. Whenever the system generates a `c` message, a buffer may be flooded. However, if the system only ever executes the cycles in which `a` and `b` messages are exchanged, the filling of every buffer remains bounded. Whether a cycle generating a `c` message ever gets executed obviously depends on how the system is initialized, which is information we have already abstracted away. Since our test is conservative and only returns a boundedness result if boundedness can be proven, i.e., there is no way of combining the cycles in the CFSM in such a fashion that the buffer can be blown up, it would in this case return an "UNKNOWN" answer.

While in the above case the boundedness can be seen by manual inspection, this is not generally the case. Consider the example given in Figure 1 in which the actual boundedness of the model is far from obvious. This calls for automated methods in support of testing a system's boundedness.

## 4 Abstracting UML RT Models

In this section we describe a sequence of conceptual abstractions for UML RT models. Each level corresponds to a computational model for which complexity results for

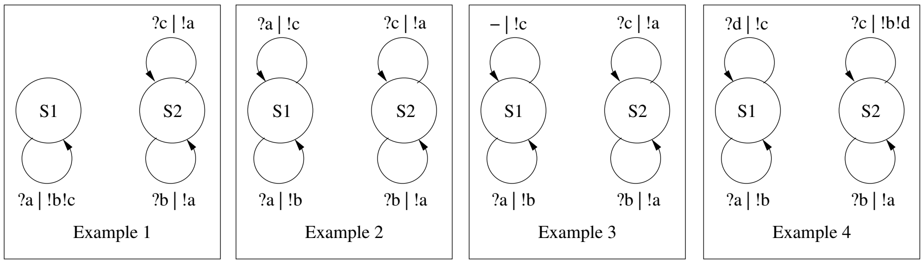


**Fig. 2.** Various Examples of Simple CFSM Systems

the boundedness problem are either known, or provided by our work. The abstraction is conceptual since the tool that we develop does not perform the transformations described in this section, but uses a more direct code analysis approach. The purpose of the conceptual abstraction is to reason about the complexity of our boundedness test. As mentioned above, we want to reason about the boundedness of the UML RT model in terms of summary message passing effects of simple control flow cycles. The goal of our conceptual abstraction is to arrive at a data structure that allows us to reason about these summary effects using linear combination analysis.

The abstract computational model that we obtain is an overapproximation of the original UML RT system in the following sense:

– All behavior of the original system is also possible in the overapproximation. However, there can exist some behavior that is possible in the overapproximation, but not in the original system.
– The abstraction preserves the number of messages in every communication channel (buffer) of the UML RT model. In particular, if some buffer is unbounded in the UML RT model, then it is also unbounded in the overapproximation. Furthermore, if a buffer is bounded by a constant $k'$ in the overapproximation, then it is bounded by some constant $k \leq k'$ in the original system.

The following summarizes the conceptual abstraction steps.

*Level 0: UML RT.* We start with the original system model described in UML RT. For the original UML RT model boundedness is, of course, undecidable, since it can contain arbitrary program code and can thus simulate Turing-machines.

*Level 1: CFSMs.* First, we abstract from the general program code on the transitions of the UML RT model and retain only the finite control structure of the capsules and their message passing via unbounded buffers representing the communication channels. We obtain a system of communicating finite-state machines (CFSMs), sometimes also called FIFO-channel systems [1]. For the CFSM model boundedness is also undecidable since CFSMs can still simulate Turing-machines [8].

*Level 2: Parallel-Composition-VASS.* In the next step we abstract from the order of the messages in the buffers and consider only the number of messages of any given type.

For example, the buffer with contents `abbacb` would be represented by the integer vector $(2, 3, 1)$, representing 2 messages of type `a`, 3 messages of type `b` and 1 message of type `c`. Also we abstract from the ability to test explicitly if a given buffer is empty. In the abstraction, all actions that are enabled when the buffer is empty are also enabled when it is non-empty.

For the purpose of complexity analysis it is helpful to relate the obtained abstraction to the theory of Petri nets. The numbers of messages in any buffer can be stored on Petri net places. We then obtain a vector addition system with states (VASS) [7]. The states correspond to the control-states of the UML RT model and the Petri net places represent the buffer contents. More exactly, we obtain a *parallel-composition-VASS*. This is a VASS whose finite-control is the parallel (but unsynchronized) composition of several finite automata. Each part of this parallel composition corresponds to the finite control of some part of CFSM of level 1, and to the finite control of a capsule in the original UML RT model. Note that a parallel-composition-VASS is not exactly the same as the parallel composition of several VASS, because the places are shared by all parallel parts of the finite control. (It will be shown later that parallel-composition-VASS are in some respects more succinct that normal VASS.) The boundedness problem for parallel-composition-VASS is polynomially equivalent to the boundedness problem for Petri nets, which is $EXPSPACE$-complete [32].

*Level 3: Parallel-Composition-VASS with Arbitrary Input.* We now abstract from activation conditions of cycles in the control-graph of the VASS and assume instead that there are always enough messages, represented by tokens, present to start the cycle. For example, a cycle that first reads one message `a` from a buffer and then writes two messages `a` to the same buffer can be repeated infinitely often, but only if in the beginning there was at least one message `a` in the buffer. Any (combination of) cycles with an overall positive effect on all places has a minimal activation condition, i.e., a minimal number of tokens needed to get it started. In principle, it is decidable if there is a reachable configuration that satisfies these minimal requirements, but this involves solving the coverability problem for VASS (i.e., Petri nets). The coverability problem is the question if there exists a reachable marking which is bigger than a given marking. This problem is decidable, but at least $EXPSPACE$-hard [22, 13], and thus not practical. Therefore we use this overapproximation and assume that these activation conditions can always be satisfied. More precisely, we assume that any cyclic sequence of transitions that has an overall non-negative effect on all places can be repeated arbitrarily often. As far as boundedness is concerned, we replace the problem 'Is the system bounded if starting at the given initial configuration?' by the problem 'Is the system bounded for any finite initial configuration?', also referred to as the *structural boundedness problem*. Obviously, every unbounded system is also not structurally bounded. It will be shown in Section 7 that this structural boundedness problem for parallel-composition-VASS is co-$\mathcal{NP}$-complete, unlike for standard Petri nets where it is polynomial [26, 13]. (The reason for this difference is that an encoding of control-states by Petri net places does not preserve structural boundedness, because it is not assured that only one of these places is marked at any time.) Furthermore, the co-$\mathcal{NP}$-lower bound even holds if the number of simple cycles in the control-graph is only linear in the size of the system description.

*Level 4: Independent Cycle System.* Finally, we abstract from the fact that certain cycles in the control graph depend on each other. For example, cycles might be mutually exclusive so that executing one cycle makes another cycle unreachable, or imply each other, i.e., one cannot repeat some cycle infinitely often without repeating some other cycle infinitely often. Instead we assume that all cycles are independent and any combination of them is executable infinitely often, provided that the combined effect of this combination on all places is non-negative. It should be noted that one part of this overapproximation condition, the mutually exclusive cycles, is normally not a problem anyway. This is because in almost all practical cases the control-graph of the capsules in UML RT models is strongly connected and therefore cycles are not mutually exclusive.

The *un*boundedness problem for this abstracted model then becomes the following question: Is there any linear combination (with positive integer coefficients) of the effects of simple cycles in the control graph, such that the combined effect is non-negative on all places and strictly positive on at least one place? Since we consider an overapproximation, the original UML RT model is surely bounded if the answer to this question is 'no'. Since these effects of simple cycles can be represented by integer vectors, we get the following problem. Given a set of integer vectors, does there exist a linear combination (with positive integer coefficients) of them, such that the result is non-negative in every component and strictly positive in at least one. This problem can be solved in time polynomial in the number of vectors by using linear programming techniques. However, in the worst case the number of different simple cycles and corresponding vectors can be exponential (in the size of the control-graph), although they are not necessarily completely independent. So far, we only have an exponential-time upper bound on the *worst-case* complexity of checking boundedness at this abstraction level 4. However, the important aspect is that the time required is only polynomial in the number of simple cycles, unlike at level 3, where the problem is co-$\mathcal{NP}$-hard even for a linear number of simple cycles. This is very significant, since for instances derived from typical UML RT models, the number of simple cycles is usually small (see Section 7).

It is easy to see that the abstraction obtained at level 4 is an overapproximation of the UML RT model in the sense defined above.

## 5 The Concrete Abstraction

We now present the concrete abstraction of UML RT models as we are currently implementing it in the tool IBOC (*IMCOS Boundedness Checker*). Due to space limitations, the presentation only sketches the approach, a complete presentation including correctness arguments will be included in a forthcoming paper.

The objective of the concrete abstraction is to automatically transform UML RT models into a collection of *effect vectors*, each of which represents the summary message passing effect of each simple cycle in each capsule state machine. In order to obtain these vectors we extract the control flow graph of each capsule state machine from the UML RT model[1]. We annotate the edges of this graph, which we call *effect graph*, with the summary message passing effect of the UML RT model transition that

[1] Note that we currently assume the transition code to only consist of linear, non-branching and non-iterating control flow structures.

it corresponds to. To obtain the effect vectors we determine the simple cycles in the effect graphs using a modified depth-first search (DFS) procedure. Figure 3 presents the effect graphs that we obtain for the 2-Capsule model in Figure 1. The analysis returns the effect vectors $v_1 = (4, 1, -2)$ and $v_2 = (-1, -1, 1)$.

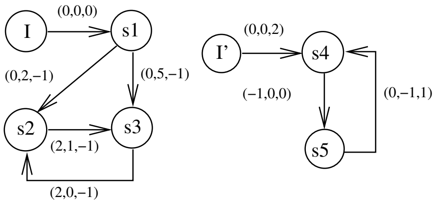


**Fig. 3.** Effect Graphs of the 2-Capsule Model from Figure 1

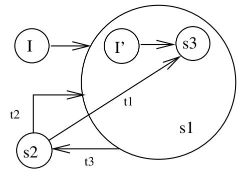


**Fig. 4.** Effect graph of Hierarchical State Machine

The above presentation refers to flat state machines, however, the effect graphs derived from UML RT state machines may be hierarchical as shown in Figure 4. The problem is compounded by group transitions that return to history, such as transition `t2` in the example. We are adjusting the DFS algorithm to deal with composite states. If a vertex $v$ corresponding to some state is enclosed by several composite vertices, we also collect all the successors of the enclosing vertices in addition to its own. That exactly resembles the behavior of the common edges which represent group transitions. When the target of an edge is a composite vertex, say $v'$, a procedure is invoked to recall the latest non-composite vertex enclosed by $v'$ in the vertex stack. The latest enclosed vertex corresponds to the last active substate when a return to history transition is taken. For the example in Figure 4, the modified DFS algorithm will determine the following cycles: $(I', t3, S2, t2, I')$, $(S2, t1, S3, t3, S2)$ and $(S3, t3, S2, t2, S3)$. Note the different targets for the return-to-history transition `t2`.

## 6 Boundedness Test

*Overall Boundedness Test.* We now describe the boundedness test using a linear combination analysis of the effect vectors that have been derived from the UML RT model. For every buffer and every message type there is one component in each of the effect vectors. The component can be negative if in the cycle more messages of this type were removed from a buffer than added to it. The resulting semilinear system is unbounded if and only if there exists a linear combination with positive coefficients of the effect-vectors that is positive in every component and strictly positive in at least one component. Formally, this can be described as follows: Let $v_1, \ldots, v_n \in \mathbb{Z}^k$ be the effect-vectors of all simple cycles and let $v^j$ be the $j$-th component of the vector $v$. The question then is

$$\exists x_1, \ldots, x_n \in \mathbb{N}_0. \sum_{i=1}^{n} x_i v_i \geq \mathbf{0} \wedge \exists j. \left( \sum_{i=1}^{n} x_i v_i \right)^j > 0.$$

This can easily be transformed into a system of linear inequations and solved by standard linear programming tools. If this condition is true then our overapproximation is unbounded, but not necessarily also the UML RT model. The unboundedness could simply be due to the coarseness of the overapproximation. On the other hand, if the condition above is false, then our overapproximation is bounded, and thus our original UML-RT model is also bounded. Thus, this test yields an answer of the form "BOUNDED" in case no linear combination of the effect vectors satisfying the above constraint can be found, and "UNKNOWN" when such a linear combination exists.

*Examples.* Consider Example 1 from Figure 2. The effect vectors are $v_1 = (-1, 1, 1)$, $v_2 = (1, 0, -1)$ and $v_3 = (1, -1, 0)$. Obviously, $x_1 = x_2 = 1$ describes a linear combination satisfying the above constrains and we conclude "UNKNOWN". In fact, Example 1 is unbounded under any initialization of the message buffers with either `a`, `b` or `c` messages. For Example 2 the vectors are $v_1 = (-1, 0, 1)$, $v_2 = (-1, 1, 0)$, $v_3 = (1, 0, -1)$ and $v_4 = (1, -1, 0)$. It is easy to see that there is no linear combination of these vectors satisfying the above constraint, hence we conclude "BOUNDED". Similarly, Examples 3 and 4 lead to results "UNKNOWN". For the 2-Capsule example of Figure 1 we had obtained the effect vectors vectors $v_1 = (4, 1, -2)$ and $v_2 = (-1, -1, 1)$. To represent the $> 0$ condition in the linear inequation system we add a constraint $3x_1 - x_2 \geq 1$. The linear inequation solver returns infeasibility of this system of inequations, and we thus conclude a result of "BOUNDED". Note that it is not easy to see this result by manual inspection

*Computing Bounds for Individual Buffers.* A more refined problem is to compute upper bounds on the lengths of individual buffers in the system. In particular, some buffers might be bounded even if the whole system is unbounded. Since normally not all buffers can reach maximal length simultaneously, the analysis is done individually for each buffer $B$. This can be done by solving a linear programming problem that maximizes a target function $f_B$. $f_B$ is a linear function whose input is our abstracted system configuration, i.e., a vector of natural numbers indicating how often which message type occurs in which buffer, and which returns the length of buffer $B$. Let $m$ be the number of capsules in the system. Let $C_0$ be the initial configuration of the system, $C_B$ the reachable configuration where buffer $B$ has maximal length, and $p$ the path from $C_0$ to $C_B$. Then $p$ can be decomposed into $m$ parts $p_1, \ldots, p_m$ such that $p_i$ is the part of $p$ that occurs in the i-th capsule. Each $p_i$ starts at control-state $s_0^i$, the initial state of capsule $i$. Each $p_i$ can be further decomposed into a part consisting of simple cycles and a non-cyclic part. The order of these is not important for us, since we are only interested in the effect-vectors of these paths. Let $p(s_0^i, s^i)$ be the non-cyclic part of $p_i$ and $s^i$ some control-state in capsule $i$. For any path $p$ let $E(p)$ be its effect-vector. Then $E(p_i) = E(p(s_0^i, s^i)) + \sum_{i=1}^{n} x_i v_i$ for some $x_i$. It follows that $E(p) = \sum_{i=1}^{m} E(p(s_0^i, s^i)) + \sum_{i=1}^{n} y_i v_i$ for some $y_i$. In order to maximize our target function $f_B$ we need to apply it to $E(p)$ and find the optimal paths to control-states $s^1, \ldots, s^m$ (note that the same control-state might be reachable via several different paths), and the optimal numbers $y_1, \ldots, y_n$. We thus need to compute $max := max(p(s_0^1, s^1), \ldots, p(s_0^m, s^m), y_1, \ldots, y_n) f_B(E(p))$.

However, the combinatorial complexity of finding the optimal paths and control-states $s^i$ in all capsules to maximize $f_B$ is too big for practical purposes since one would need to try out all exponentially many possible combinations. Therefore, we apply yet another overapproximation to simplify the problem. Let $r_i$ be the minimal vector s.t. $\forall s^i.\ r_i \geq E(p(s_0^i, s^i))$. In other words, we maximize individual components of vector $E(p(s_0^i, s^i))$ over all paths to control-states $s^i$. The vectors $r_i$ can easily be computed in polynomial time by depth-first search in the individual capsules. Then we define $max' := max(y_1, \ldots, y_n) f_B\,(\sum_{i=1}^{m} r_i + \sum_{i=1}^{n} y_i v_i)$ which requires just one instance of a linear programming problem to be solved. It is easy to see that $max' \geq max$ and thus we have computed an upper bound. Normally, the function $f_B$ will be the number of messages in some buffer $B$, but it can be any linear function. For example, one might want to count only some types of messages in a buffer and not others.

*Example.* Having established boundedness of the 2-Capsule example of Figure 1, we now compute the estimated upper bound for each buffer (port). First we compute the effect vectors for all non-cyclic paths. They are listed in Table 1 where $init$ and $init'$ are the initial states of the state machines. Then we take the maxima of individual components from those effect vectors and construct the overapproximated maximal effect vectors for capsule `Left` as $r_1 = (2, 5, 0)$ and for capsule `Right` as $r_2 = (0, 0, 2)$. Thus the sum is $\sum_{i=1}^{n} r_i = (2, 5, 2)$. We obtain the following two optimization problems (1-4 and 5-8) for the two buffers left-to-right and right-to-left:

$$max : 2 - 2x_1 + x_2 \quad (1) \qquad max : 7 + 5x_1 - 2x_2 \quad (5)$$
$$2 + 4x_1 - x_2 \geq 0 \quad (2) \qquad 2 + 4x_1 - x_2 \geq 0 \quad (6)$$
$$5 + x_1 - x_2 \geq 0 \quad (3) \qquad 5 + x_1 - x_2 \geq 0 \quad (7)$$
$$2 - 2x_1 + x_2 \geq 0. \quad (4) \qquad 2 - 2x_1 + x_2 \geq 0. \quad (8)$$

Linear Programming returns a value of 6 for the objective function (1) and a value of 18 for the objective function (5). These values represent the estimated bounds for the communication buffers associated with `port1` and `port2`, respectively.

| The non-cyclic path | The effect vectors | The non-cyclic path | The effect vectors |
|---|---|---|---|
| $< init, s1 >$ | (0,0,0) | $< init, s1, s2 >$ | (0,2,-1) |
| $< init, s1, s2, s3 >$ | (2,3,-2) | $< init, s1, s3 >$ | (0,5,-1) |
| $< init, s1, s3, s2 >$ | (2,5,-2) | $< init', s4 >$ | (0,0,2) |
| | | $< init', s4, s5 >$ | (-1,0,2) |

**Table 1.** The Effect Vectors for all Non-Cyclic Paths in 2-Capsules

## 7 Complexity Analysis

In this Section we analyze the complexity of the problem of checking for boundedness in general and our algorithm in particular. It has already been mentioned in Section 3 that the boundedness problem is undecidable for UML RT (level 0), undecidable for CFSMs (level 1), $EXPSPACE$-complete for VASS (level 2), co-$\mathcal{NP}$-complete for VASS with arbitrary input (level 3), and polynomial in the number of simple cycles for Independent Cycle Systems (level 4). The only part that still needs to be shown is the $\mathcal{NP}$-completeness for parallel-combination-VASS with arbitrary input.

STRUCTURAL BOUNDEDNESS OF PARALLEL-COMPOSITION-VASS

**Instance:** A VASS whose finite control is given as an unsynchronized parallel composition of automata $G_1 \| \ldots \| G_n$.

**Question:** Is the system structurally bounded, i.e., is it bounded for every initial configuration?

**Lemma 1.** *Structural boundedness of parallel-composition-VASS is co-$\mathcal{NP}$-hard, even if all the control-graphs $G_i$ are strongly connected and contain only polynomially many simple cycles.*

*Proof.* We reduce SAT to unboundedness of parallel-composition-VASS for some initial configuration. Let $\Phi := Q_1 \wedge \ldots \wedge Q_k$ be a boolean formula over variables $x_1, \ldots, x_n$. Each clause $Q_j$ is a disjunction of literals and each literal is either a variable or the negation of a variable. We now construct in polynomial time a parallel-composition-VASS as follows. The system contains $k+2$ places, $p_1, \ldots, p_k, l, g$, where the first $k$ places each correspond to one clause, and places $l, g$ have special functions.

For every variable $x_i$ we define an automaton $G_i$ with three states $s_i, t_i, f_i$ and the following transitions. We describe the effects of these transitions on the places by vectors of integers, as usual in VASS. There are transitions from $s_i$ to $t_i$, $t_i$ to $s_i$, $s_i$ to $f_i$ and $f_i$ to $s_i$ that each have the following effect. They reduce place $l$ by 1 and leave all other places unaffected. There is a transition from $t_i$ to $t_i$ with the following effect: For all $j$, if clause $Q_j$ contains literal $x_i$ then one token is added to $p_j$. Furthermore, exactly one token is removed from place $g$. There is a transition from $f_i$ to $f_i$ with the following effect. For all $j$, if clause $Q_j$ contains literal $\neg x_i$ then one token is added to $p_j$. Furthermore, exactly one token is removed from place $g$.

Finally, we add another automaton $T$ with just the state $s$ and a transition from $s$ to $s$ with the effect $(-1, \ldots, -1, 0, n+1)$. We now show that the VASS with these places and finite control $G_1 \| \ldots \| G_n \| T$ and initial control-state $(s_1, \ldots, s_n, s)$ is structurally bounded iff $\Phi$ is not satisfiable[2].

If $\Phi$ is satisfiable then there exists a variable assignment that makes all clauses $Q_j$ true. Then there exists an unbounded run of the system of the following form. If $x_i$ is true then go from $s_i$ to $t_i$, else go to $f_i$. The combined effect of this is $(0, \ldots, 0, -n, 0)$. Then do each local cycle at $t_i$ (resp. $f_i$) exactly once. The combined effect of this is $(e_1, \ldots, e_k, 0, -n)$, where for all $j$ we have $e_j \geq 1$. Then we do the cycle at $s$ exactly once. The effect of this is $(-1, \ldots, -1, 0, n+1)$. Thus the combined effect is $\geq (0, \ldots, 0, +1)$. This combination of cycles can then be repeated infinitely often. Thus there exists an unbounded run starting at configuration $(0, \ldots, 0, n, n)$. So the system is not structurally bounded.

Now assume that $\Phi$ is not satisfiable. No infinite run from any initial configuration can change infinitely often between some $s_i$ and $t_i$/$f_i$, because place $l$ is decreased in these transitions and never increased anywhere else. Thus, for every $i$, every infinite run can only contain infinitely many loops at $t_i$ or at $f_i$ but not infinitely many of both. By the construction of the automata $G_i$, and since there is no satisfying assignment for $\Phi$, no combination of these loops can have a strictly positive effect on all places $p_1, \ldots, p_k$. Therefore, for any initial configuration, the loop at $s$ can only be done finitely often. Therefore, the local loops at states $t_i$/$f_i$ can only be done finitely often, because of their effect on place $g$. Thus all runs from any initial configuration have finite length and the system is structurally bounded. □

[2] Note also that each $G_i$ and $T$ are strongly connected and that the total number of simple cycles in the system is $4n+1$.

**Lemma 2.** *Structural boundedness of parallel-composition-VASS is in co-$\mathcal{NP}$.*

For the proof we refer the reader to the full version of the paper. We conclude the following theorem:

**Theorem 3.** *Structural boundedness of parallel-composition-VASS is co-$\mathcal{NP}$-complete.*

The co-$\mathcal{NP}$-hardness of the structural boundedness problem at abstraction level 3, even for small numbers of simple cycles, justifies for further abstraction to level 4, where the problem is polynomial in the number of simple cycles.

To analyze the complexity of our boundedness test algorithm for UML RT models, consider a typical input system. It consists of $m$ capsules running in parallel and communicating with each other via buffers. Let $k$ be the maximal size of each buffer. Thus the size of the instance is $n := \mathcal{O}(m * k)$. Note that the total number of different control-state combinations is $\mathcal{O}(k^m)$, the classical state explosion problem. However, our algorithm avoids this combinatorial explosion.

First, it extracts (the effects of) all simple cycles from the finite controls of each capsule. The time needed for this is polynomial in the number of simple cycles. Then it checks for the existence of positive linear combinations of the effects of these cycles. Again, the time required is polynomial in the number of simple cycles (by using linear programming techniques). Thus, the algorithm overall requires polynomial time in the number of simple cycles.

In the worst case, the number of simple cycles in any capsule (of size $k$) can be exponential in $k$, i.e., $\mathcal{O}(2^k)$. So the total number of simple cycles in the system is only bounded by $\mathcal{O}(m * 2^k)$. Thus the worst-case complexity of the algorithm is $\mathcal{O}(poly(m * 2^k))$. It should be noted that this is still normally much smaller than $\mathcal{O}(2^{m*k}) = \mathcal{O}(2^n)$. However, these worst-case complexity estimates are not very meaningful for practical problems. In typical UML RT models the finite-control graphs in the capsules are derived from programming-language-like control-flow graphs. These graphs are normally very sparse, and the number of simple cycles in them is normally polynomial, rather than exponential. Therefore, for typical UML RT models, the algorithm requires only polynomial time.

## 8 Experimental Results

We now report on experiments that we performed using the IBOC system. We used the 2-Capsule model of Figure 1, a UML RT model of the Alternating Bit Protocol, and the UML RT model of a telecommunications switch, called PBX. For experimentation purposes we obtained the PBX model from IBM/Rational. It is a model with a complexity comparable to that of models used in industrial development projects.

IBOC directly accesses the internal model structure inside the Rose RealTime tool and uses the LPSOLVE system for linear programming tasks. Table 2 shows the performance results of these experiments that are performed on a two processor 1GHz Pentium III PC with 2 GB memory.

The IBOC system returned "precise" boundedness results in the sense that an "UNKNOWN" verdict in all cases corresponded to an unboundedness in the respective UML

RT model. For the model of Alternating Bit protocol, for instance, IBOC returned "UNKNOWN" and provided two counterexamples as linear combinations of cycles that potentially contribute to the unbounded growth of channels. These counterexamples indicate that two cycles in the state machine of the sender capsule may cause the unboundedness. This result is plausible since the sender injects messages into the Alternating Bit system without restraint. The PBX model is obviously of a complexity that makes it impossible to guess boundedness with manual methods. IBOC returns a "BOUNDED" result within very reasonable runtime, which proves that our analysis scales to UML RT models of realistic size. To assess the quality of the estimated buffer bounds we executed the PBX model in Rose RealTime and traced several ports. For most ports, the actual bounds are very close to the estimates. For instance, a port $configureDialPlan$ is observed to contain no more than five messages at runtime, while the estimate is seven.

| | 2-Capsule | Alternating Bit | PBX |
|---|---|---|---|
| Checked capsules | 3 | 4 | 29 |
| Checked states | 30 | 47 | 736 |
| Checked transitions | 8 | 15 | 299 |
| Checked message types | 3 | 8 | 308 |
| Checked buffers | 2 | 4 | 57 |
| Reported cycles | 3 | 11 | 2030 |
| Generated vectors | 2 | 11 | 1026 |
| Runtime for cycle detection [sec.] | 0.034 | 0.136 | 24.860 |
| Runtime for boundedness check [sec.] | 0.233 | 1.110 | 28.110 |
| Runtime for computing bounds [sec.] | 0.207 | - | 3.250 |

**Table 2.** Experimental Results obtained with the IBOC System

## 9 Related Work

There is a vast body of literature on the problem of dealing with the unboundedness of communication queues. This includes overapproximations using lossiness assumptions for queues [1] (the boundedness problem stays undecidable for lossy queue systems [2], even under strong restrictions [24]), sufficient syntactic conditions for the unboundedness of communication channels in CFSM systems [21], the symbolic treatment of infinite queues [5, 6] and the elimination of unbounded queues using structural properties of the state spaces of the CFSMs [10].

At the time of writing, no operational semantics for UML RT is available. Work described in [14] and [15] focuses on giving semantics to the structural aspects of UML RT. The translation of UML RT models into Promela was fist attempted by [27] which pointed out the tremendous problems involved in dealing with UML RT queues and their potential unboundedness. Our analysis of hierarchical UML RT CFSMs is in part based on ideas from [3].

Model Checking based on integer inequality systems has been pioneered by the INCA system [9]. Esparza and Melzer used integer linear programming to check several safety properties (e.g., mutual exclusion) for Petri nets models [25, 12]. However, in most cases, the models considered were 1-safe Petri nets which are bounded by definition.

## 10 Conclusion

We presented an incomplete test for the boundedness of communication buffers in UML RT models. Our algorithm abstracts UML RT models such that only the communication effects of the simple control flow cycles in the capsule state machines remain. The test then tries to establish a linear combination of the resulting effect vectors that allows at least one of the system's message buffers to grow unboundedly. If such a linear combination cannot be found, the system is bounded. In addition we proposed an upper bound estimate for the maximal occupancy of individual buffers. We have argued that our analyses scale well to UML RT systems of realistic complexity, and supported this claim by experimental results using the IBOC tool.

One focus of current research is to refine the analysis, in particular when the result is "UNKNOWN". The IBOC system that we currently develop permits the identification of a sub-model to which the boundedness analysis can be limited. Another focus lies on enhancing the generation of "counterexamples", i.e., sets of cycles that lead to unboundedness. We are also interested in developing abstraction refinement procedures when the counterexamples are spurious, i.e., not executable in the original UML RT model. Future work will extend the analysis to establish boundedness results for more general types of dynamic systems, e.g., systems that dynamically generate and delete concurrent processes, or systems that dynamically allocate and deallocate objects on heap memory. Boundedness in these cases implies the absence of memory leaks due to improper memory management.

*Acknowledgements.* We thank John Hogg, Andreas Podelski and Bran Selic for initial discussions on the subject of this paper. IBM/Rational supported this work by providing licenses for the Rational Rose RealTime tool. The third author was supported through the DFG funded project IMCOS (grant number LE 1342/1).

## References

1. P. Abdulla and B. Jonsson. Verifying Programs with Unreliable Channels. In *LICS'93*. IEEE, 1993.
2. P. Abdulla and B. Jonsson. Undecidable verification problems for programs with unreliable channels. *Information and Computation*, 130(1):71–90, 1996.
3. R. Alur, R. Grosu, and M. McDougall. Efficient reachability analysis of hierarchical reactive machines. In *Proc. of CAV'00*, volume 1855 of *LNCS*. Springer Verlag, 2000.
4. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison Wesley, 1998.
5. B. Boigelot and P. Goidefroid. Symbolic verification of communication protocols with infinite state spaces using qdds. In *Proc. CAV'96*, volume 1102 of *LNCS*. Springer, 1996.
6. A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. In *Proc. of ICALP'97*, volume 1256 of *LNCS*, 1997.
7. A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In *Proc. of STACS'99*, volume 1563 of *LNCS*. Springer Verlag, 1999.

8. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 2(5):323–342, April 1983.
9. James C. Corbett and George S. Avrunin. Using integer programming to verify general safety and liveness properties. *Formal Methods in System Design: An International Journal*, 6(1):97–123, January 1995.
10. W. Damm and B. Jonsson. Eliminating queues from rt uml models. In *Proc. of FTRTFT 2002*, LNCS. Springer, 2002.
11. L.E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with distinct factors. *American Journal of Mathematics*, 35:413–422, 1913.
12. J. Esparza and S. Melzer. Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods in System Design*, 16:159–189, 2000.
13. J. Esparza and M. Nielsen. Decibility issues for Petri nets - a survey. *Journal of Informatik Processing and Cybernetics*, 30(3):143–160, 1994.
14. C. Fischer, E.-R. Olderog, and H. Wehrheim. A csp view on uml-rt structure diagrams. In *Fundamental Approaches to Software Engineering, Proc. of the 4th International Conference, FASE 2001*, volume 2029 of *LNCS*. Springer Verlag, 2001.
15. R. Grosu, M. Broy, B. Selic, and G. Stefanescu. What is behind UML-RT? *Behavioral specifications of businesses and systems*, 1999.
16. E. M. Gurari and O. H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *Journal of Computer and System Sciences*, 22:220–229, 1981.
17. D. Harel. Statecharts: A visual formalisation for complex systems. *Science of Computer Programming*, 8:231–274, 1987.
18. D. Herzberg and A. Marburger. The use of layers and planes for architectural design of communication systems. In *Proc. of the Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing ISORC 2001*. IEEE Computer Society, May 2001.
19. Gerard J. Holzmann. *The Spin Model Checker - Primer and Reference Manual*. Addison-Wesley, 2004.
20. O. Ibarra. Reversal-bounded multicounter machines and their decision problems. *Journal of the ACM*, 25:116–133, 1978.
21. T. Jeron and C. Jard. Testing for unboundedness of fifo channels. *Theoretical Computer Science*, (113):93–117, 1993.
22. R. Lipton. The reachability problem requires exponential space. Technical Report 62, Department of Computer Science, Yale University, January 1976.
23. A. Lyons. Developing and debugging real-time software with objectime developer. available from `http://www.objectime.com/otl/technical/1999q1_p017.pdf`, 1999.
24. R. Mayr. Undecidable problems in unreliable computations. *TCS*, 297(1-3):337–354, 2003.
25. S. Melzer and J. Esparza. Checking system properties via integer programming. In H.R. Nielson, editor, *Proc. of ESOP'96*, volume 1058 of *Lecture Notes in Computer Science*, pages 250–264. Springer Verlag, 1996.
26. G. Memmi and G. Roucairol. Linear algebra in net theory. In *Net Theory and Applications*, volume 84 of *LNCS*, pages 213–223, 1980.
27. M. Saaltink. Generating and analysing Promela from RoseRT models. Technical Report TR-99-5537-02, ORA Canada, 1208 One Nocholas Street, Ottawa Ontario, K1N 7B7, Canada, 1999.
28. B. Selic. Turning clockwise: using UML in the real-time domain. *Comm. of the ACM*, 42(10):46–54, Oct. 1999.
29. B. Selic. An overview of uml 2.0. International Conference on Software Engineering, Tutorial Notes, May 2003.

30. B. Selic, G. Gullekson, and P.T. Ward. *Real-Time Object-Oriented Modelling*. John Wiley & Sons, Inc., 1994.
31. B. Selic and J. Rumbaugh. Using UML for modeling complex real-time systems. http://www.rational.com/media/whitepapers/umlrt.pdf, March 1998.
32. H. Yen. A unified approach for deciding the existence of certain Petri net paths. *Information and Computation*, 96(1):119–137, 1992.

# A Appendix

The material in this appendix is only included for the benefit of the reviewers. The proof contained herein will not be included in the final version of the paper.

**Lemma 4.** *Structural boundedness of parallel-composition-VASS is in co-$\mathcal{NP}$.*

*Proof.* We reduce the negation of the problem, unboundedness for some initial configuration, to the language-non-emptiness problem for reversal-bounded counter machines. These are Minsky-multi-counter machines where the counters can only switch from increasing-mode to decreasing-mode (and vice versa) a bounded number of times [20]. The reversal-bounded counter machine $C$ is constructed from our parallel-composition-VASS as follows.

Let $G_1\| \dots \|G_m$ be the control-graph of the VASS. First, partition each $G_i$ into strongly connected components. This can be done in polynomial time. Choose nondeterministically a strongly connected component $A_i$ in every $G_i$. Choose nondeterministically some state $s_i^0$ in $A_i$ as new initial state. Let $s_1^0$, the initial state of $A_1$, be the initial control-state of our new reversal-bounded counter machine. For every index $i$ we add an extra (unlabeled) transition from $s_i^0$ to $s_{i+1}^0$. Let $d$ be the dimension of the VASS, i.e, the number of places. We now add $2 * d$ reversal-bounded counters to the machine, s.t. for every component $j$ in our vectors in $\mathbb{Z}^d$ on the transition arcs there are two counters $c_j^+$ and $c_j^-$. These counters are initially $0$. Every time an arc labeled with vector $v$ is traversed, the counters are changed as follows. For all components $j$ of the vector $v$ (with $1 \leq j \leq d$) do the following: If $v_j \geq 0$ then add $v_j$ to $c_j^+$. If $v_j < 0$ then add $-v_j$ to $c_j^-$. If follows that $c_j^+ - c_j^-$ represents the complete effect of the traversed path on component $j$. Finally, we add an extra control-state $s$ and a transition from $s_m^0$ to $s$. At state $s$ the reversal-bounded counter machine checks if $\forall j.\, c_j^+ \geq c_j^-$ and if there exists some $j$ s.t. $c_j^+ > c_j^-$. It accepts iff these conditions are satisfied. This counter machine has polynomial size, uses a polynomial number of counters which are all reversal-bounded by 1 (i.e., only make one change from increasing to decreasing mode). The one reversal is needed for the final check at the end, because one has to subtract $c_j^-$ from $c_j^+$. Checking the language of $C$ for non-emptiness is in $\mathcal{NP}$, as shown by Ibarra [16]. This the whole algorithm works in nondeterministic polynomial time.

It remains to show that the parallel-composition-VASS is not structurally bounded iff the language of $C$ is non-empty.

The parallel-composition-VASS is not structurally bounded iff there exists some initial configuration from which there is an infinite unbounded run. By Dickson's Lemma [11] this is the case iff it has some infinite unbounded and ultimately periodic run. The period of this run must have an overall positive effect on all places and a strictly positive effect on at least one place. Furthermore, in the control-graph this period forms a large cycle $cy$, which can be decomposed into $m$ smaller cycles $cy_1\| \dots \|cy_m$ where each $cy_j$ happens in the automaton $G_j$. Obviously, each cycle $cy_j$ must be contained in some strongly connected component $A_j$ of $G_j$. By the construction of $C$ this exists if and only if $C$ can reach some accepting state. □