Using the mutual exclusion protocol for safety in the area of robotic surgery

Luzie Schreiter[1], Ammar Mohammed[2], Joerg Raczkowsky[1] and Heinz Woern[1]

[1]Institute for Process Control and Robotic, Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany
[2]Department of Computer Science, Institute of Statistical Studies and Researches, Cairo University, Egypt
luzie.schreiter@kit.edu, ammar@cu.edu.eg, joerg.raczkowsky@kit.edu, woern@kit.edu *

Abstract. A Surgical robotic system is a computer assisted electro-mechanical device which helps the surgeon to perform interventions. It is a safety critical system as it has to take in consideration the safety of patients that can be violated due to technical problems. To respect the safety property some methods are needed to guarantee the correctness of the surgical system adherence to crucial safety requirements. For this reason, formal modeling and specification are helpful. This paper shows how to model and specify the coordination between safety critical surgical robots using hybrid automata. The approach allows formal system specification on different levels of abstraction on one hand, and the expression of real-time behavior of the system with continuous variables on the other hand. The semantics of hybrid automata allows us to analyze the robotics behavior using simulation or model checking.

1 Introduction

Specifying behaviors of multi-agent systems (MASs) – also called multi-robot systems – is a demanding task, especially when applied in safety-critical systems. Where it has to be carried out carefully in order to avoid side effects that might cause unwanted or even disastrous behaviors. Thus, formal methods based on mathematical models of the system under design are helpful. They not only allow us to formally specify the system at different levels of abstraction, but also to verify the consistency of the specified systems before implementing them. Formal specification aims at providing a precise and unambiguous description of the behavior of MASs, whereas verification aims at proving the satisfaction of specified requirements. Formal methods are used in different (safety critical) areas such as air traffic control [1], vehicle control and software development [2].

A behavior of an agent can be described as discrete changes of its states with respect to external or internal actions. Whenever an action occurs, the agent moves from one

* This work has been funded by the European Commissions Seventh Framework program within the project Active Constraints Technologies for Ill-defined or Volatile Environments (ACTIVE) under grant no. 270460.

state to another. Therefore, an efficient way to model this type of discrete behaviors is to use a state transition diagrams such as a finite automaton. One remarkable advantage of transition diagrams is that they lend themselves to formal analysis techniques using *model checking*.

In realistic physical environments, however, it is necessary to consider continuous behaviors in addition to discrete behaviors of MASs. Examples of those type of behaviors include the movement of a soccer agent to kick off or to go to the ball, the process of putting out the fire by a fire brigade agent in a rescue scenario, or any other behaviors that depend on any timed physical law. The traditional state transition diagrams are not sufficient to combine these types of behaviors. *Hybrid automata* [3] offer an elegant method to capture such types of behaviors. They extend regular state transition diagrams with methods that deal with continuous actions such that the state transition diagrams are used to model the discrete changes of behaviors, while differential equations are used to model the continuous changes. The semantics of hybrid automata make them accessible to formal verification by means of model checking [4]. For the automatic verification purposes, several model checking tools for hybrid automata available are [5–7].

Computer and Robot Assisted Surgery (CRAS) systems are highly safety critical multi- agent systems. Safety for the patient and for the medical staff is one of the most important requirements during the design phase of such an adaptive system. Furthermore, an increasing number of devices in the operation theatre make the environment changing with an abundance of modular devices and sensor data. The definition of safety in context of CRAS is not trivial at all. In our understanding correctness, reliability and dependability are quality indicators for the safety of CRAS systems. The use of formal methods in the area of CRAS is quite new. In [8] the authors used Hybrid Input/Output automata for the verification of robotic surgical system. Component related invariant were used to prove an overall safety property. In [9] the authors present the application of autonomous surgery and verification of the surgical plan for a simple puncturing. In [10] the authors proved a control algorithm for directional force feedback in a surgical context. The approach shows the use of formal methods in the surgical context and its importance. This paper uses hybrid automata to model a particular scenario between two surgical robots in a shared environment. This is achieved by modeling the mutual exclusion between two robots. Some formal analysis of particular requirements on both simulation and model checking level are also shown.

The paper is structured as follows: Section 2 gives a theoretical background of Hybrid automata. In section 3 the CRAS case study is defined and the used hybrid automata are explained in detail. In section 4 the used formal analysis tools (split in simulation and model checking techniques) according to the case study are described.

2 Hybrid Automata

A hybrid automaton [3] is represented graphically as a state transition diagram, augmented with mathematical formalisms on both transitions and locations (see Fig. 2). Formally speaking, a hybrid automaton is defined as follows:

Definition 1 *A hybrid automaton is a tuple $H = (X, Q, Inv, Flow, E, Jump, Reset, Event, Init)$ where:*

- *$X \subseteq \mathbb{R}^n$ is a finite set of n real-valued variables that model the continuous evolutions of the automaton wrt. time.*
- *Q is a finite set of control locations.*
- *$Inv(q)$ is the invariant predicate, which assigns a constraint on variables X for each control location $q \in Q$. The control of a hybrid automaton remains at a location $q \in Q$, as long as $Inv(q)$ holds.*
- *$Flow(q)$ is the flow predicate on variables X for each control location $q \in Q$, which defines how the variables in X evolve over the time at location q. It constrains the time derivative of the continuous part of the variables at location q. Basically, we represent the flow as a constraint relation of the real variables to the time. In the graphical representation, a flow of a variable x is denoted as $\dot{x}$. A hybrid automaton is classified according to the type of the flow to several classes include timed,linear, rectangular and non-linear automaton.*
- *$E \subseteq Q \times Q$ is the discrete transition relation over the control locations. Each edge $e \in E$ is augmented by the following annotations:*

  Jump: *jump condition (guard), which is a constraint over X that must hold to fire transitions. Omitting a jump condition on a transition means that the jump condition is always true and it can be taken at any point of time.*

  Reset: *is a constraint, which may reset the variables by executing a specific assignments. In the graphical representation, $X' = v$ denotes that the variable X is reset with the value v. Resetting variables are omitted on transition, if the values of the variables do not change before the control goes from a location to another location.*

  Event: *synchronization label, used to synchronize and coordinate concurrent automata. These synchronization labels define the composition of the automata.*
- *$Init$ is the initial condition that assigns initial values to the variables X to each control location $q \in Q$.*

Informally speaking, the semantics of a hybrid automaton is defined in terms of a labeled transition system between states, where a state consists of the current location of the automaton and the current valuation of the real variables. To model larger systems, a hybrid automaton models each part of a system, and the communication between the different parts may occur via shared variables and synchronization labels. The behavior of the overall system is controlled using the so-called *Parallel composition*.

3 Surgical Robots Scenario

The setup under study is a reactive surgical multi-agent systems consisting of four main components: two surgical robots operating in a shared area, controller and scheduler (see Fig. 1). The main goal of the system is to provide coordination between the two robots in the shared area and to avoid collisions that might occur between the robots when they enter into the shared area phase. Additionally, the system provides a cooperation between the two robots and a controller in such a way that both robots read input from the environment and send the results to single processor in the controller. To achieve this purpose, a scheduler (a part of the controller) coordinates the behaviour of both robots when they request to send the data to the controller.

In order to prevent the robots from collision in a critical shared area, a part of the two robots should model a mutual exclusion protocol based on Fisher [11]. Also, each robot repeatedly reads its environment and sends the result to the controller. Both robots share a single processor with the controller.

In our Scenario, the scheduler coordinates the reading of both robots. In case both robots require reading processes at the same time, the scheduler ensures that the second robot has priority over the first one. That means that the second robot can interrupt the first robot reading process. As soon as the second robot finishes its reading, the first robot will reschedule its reading process.

Generally, after sending its reading, each robot delays certain time to operate its environment and to mutually coordinate the shared working area before starting a new reading process. If the result of the reading process is not sent to the controller in adequate time, the robot takes in consideration to send it again.

The role of the controller in the scenario is to coordinate the readings of the two robots using suitable communication messages. However, the communication can not be computed unless any of the two robots readings is received within at certain time (at most 10 milliseconds in our scenario). If there is a reading needed to be sent to the controller, in turn, the controller takes between 3.5 to 5.6 milliseconds to the commands. Additionally the controller acknowledges robots readings. To achieve this purpose, it needs between 0.9 and 1.0 milliseconds. In the following subsection, we model our Scenario using linear hybrid automata.

3.1 Modeling the Scenario

The scenario is modeled using linear hybrid automata. The automata communicate with each other through four variables (see Fig. 1). Since the model include two robots the index *i* represents the number of each robot.

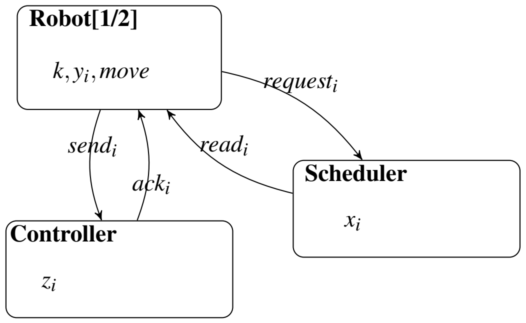

Fig. 1. Used components of the modeled scenario. Shared variables are beside the arrows and internal used variables are inside the rectangles.

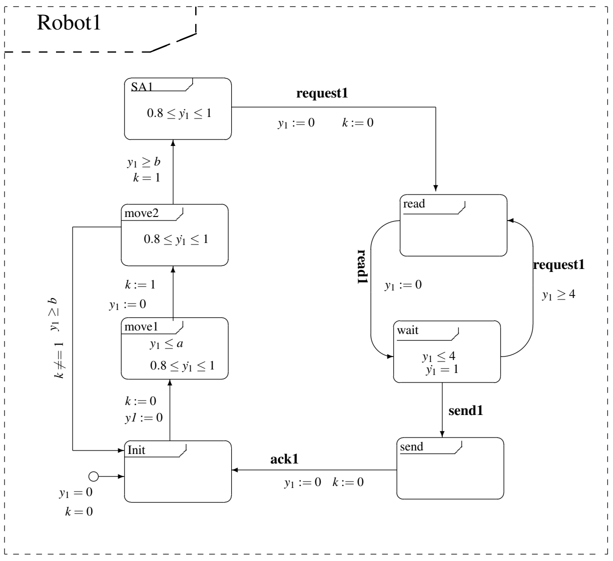

Fig. 2. Hybrid Automaton - Robot

The robot behavior The behaviors of the two robots (see Fig.2) are identical in the structure except that each robot$_i$ uses a continuous variable y_i which is used to measure the time constraints inside locations. Both robots coordinate their behavior using a shared variable k that controls mutually the robots to enter the shared area. The robot$_i$ has seven locations: **Init**, **move**$_1$, **move**$_2$, **SA**, **read**, **wait**, and **send**. At the beginning of the execution, the control of the robot$_i$ is in location **Init** with $y_i := 0$ and $k := 0$. In the location **move**$_1$, the robot has to wait no more than a constant time unit a to update the value of the variable k, i.e. assigning $k := i$. In the location **move**$_2$, the robot waits a lower time bound delay b before accessing the shared location **SA**. After remaining in the latter location for a sufficient delay, the robot starts to initiate a reading request using the event *request$_i$* and resets y_i and k to zero. In the location *read*, the robot waits until *read$_i$* event occurs from the scheduler in order to prepare the reading. As soon as the reading is allowed, the control goes to **wait** location. The robot remains in the location **wait** for up to 4 time units. During this time, the robot is ready to send its reading to the controller using *send$_i$* event. This event takes place as soon as the controller is ready to receive the reading. If the robot did not receive *send$_i$* event in time, it initiates a new reading request and goes back to location *read*. After sending the reading, the robot waits in location **send** for acknowledgment from the controller using *ack$_i$* event, and then it moves back to location **Init** and resets $y_i := 0$ and $k := 0$.

The scheduler The Scheduler automaton (see Fig. 3) is responsible for allocating each robot CPU time on a shared processor. For this purpose, it uses the event *request$_i$* to synchronize the two robots. This event indicates that the robot$_i$ is initiating a request for CPU time to prepare a reading using the event *read$_i$*. The automaton uses two contin-

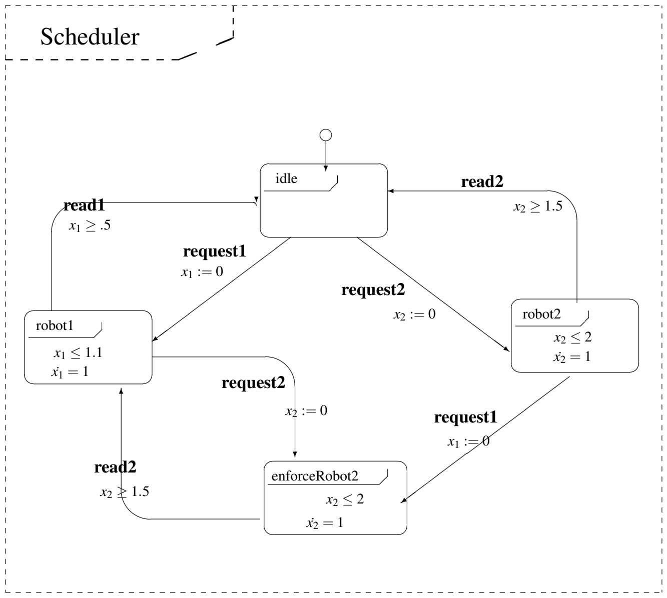

Fig. 3. Hybrid Automaton - Scheduler

uous variables x_1 and x_2 to model the time that robot$_1$ and robot$_2$ have received since the last request. The first robot takes between 0.5 and 1.1 milliseconds for the process of reading the environment, while the second robots needs between 1.5 and 2.0 milliseconds. The automaton has four locations: **idle**, **robot**$_1$, **robot**$_2$, and **enforceRobot2**. These locations are responsible for the variations of requests. Initially, the control is in location **idle** for sending a signal from the robot. The control goes to location **robot**$_1$ or **robot**$_2$, when each robot requests a reading. The control locations goes to **enforcerRobot2**, when both robots have pending requests. In case of both robots requesting the CPU, a priority is given to robot$_2$.

The Controller The Controller automaton (see Fig. 4) uses a variable z to control the behavior. initially, the control is in location **idle** for sending a signal from the robot. The signal is acknowledged and the location **wait**$_1$ or **wait**$_2$ is entered. In the previous location, the automaton has to wait for sending signal up to 10 time units. If $signal_2$ event, or respectively $signal_1$, does not occur in time, the invariant $z \leq 10$ forces the control to return to **idle** location using $expire_1$ or $expire_2$. If *send* event occurs in time, it is acknowledged and the control goes to location **compute** in which the controller computes the time required to calculate the command to send to the robot. Additionally, the controller is augmented with a clock variable c which is used to measure the time elapsed since the last signal has been sent to the robot or since the run has begun when there was no signal sent.

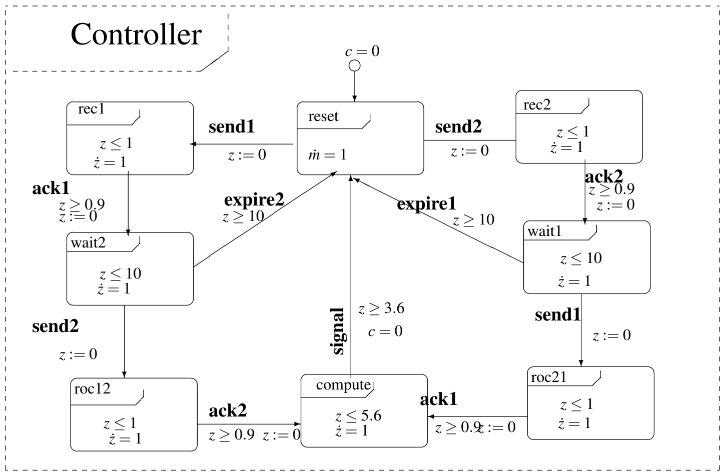

Fig. 4. Hybrid Automaton - Controller

4 Formal Analysis

Having defined and modeled our scenario using hybrid automata, one can analyze the behaviors of the model using either simulation or formal verification tools. For this purpose, there are many of tools and languages that have been proposed over the years to analyze the behavior of hybrid systems. Simulation tools, for example [12], analyze that have been proposed using several runs of the model. Simulation is powerful to analyze more continuous dynamics. Formal verification tools on the other hand are appealing as a concept to guarantee design correctness, but are limited to less continuous dynamics. In this section we first investigate the formal analysis based on the simulation. A step further, we exploit the formal analysis of Scenario with the help of model checking; in particular using one of standard model checkers of hybrid automata HyTech [5].

4.1 Simulation

We simulate the described scenario in Matlab Simulink Statecharts. Therefore we used two lightweight robot models and modeled the critical structure as a sphere. Both robots are not allowed to enter the critical structure at the same time. Since the second robot is prioritized, the critical structure is entered only from this robot if both robots are requested to enter, see Fig. 5. The notation of statecharts allows the precise specification of state-based systems. Hence it is possible to show the fundamental behavior of the case study. But to prove other system requirements, especially time specification, it is necessary to use other verification techniques. Therefore we decided to use model checking tools, which are described in the following sub-section.

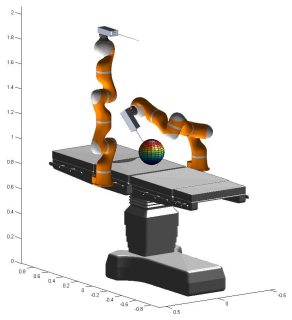

Fig. 5. Simulation - Robot 2 is entering the critical structure

4.2 Model Checking

As we already mentioned, hybrid automata are equipped with a formal semantics, making it possible to apply not only simulation but also formal methods in order to prove certain properties of the specified systems, e.g. by model checking. However, in the context of hybrid automata, the term *model checking* usually refers to *reachability* testing, i.e. the question whether some state is reachable from the initial configuration of the specified model. A model checker first computes the reachability and then checks the reachability of a property of interest by simply intersecting the sets of reachable states and the property.

For the behavior specification of our case scenario, we can conduct several experiments using the standard model checker HyTech [5]. HyTech is implemented for the analysis of linear hybrid automata. They take textual representations of hybrid automata like the one in Fig. 6 as input and perform reachability tests on the state space of the resulting product automaton. This is usually done by first computing all reachable states from the initial configuration, and then checking the resulting set for the needed properties. Following the specification formalism RCTL [13], a reachability of the property Ψ asserts that starting from an initial state, there is a region along a run in which Ψ is satisfiable. This can be specified in RCTL as follows:

$$init \rightarrow \exists\Diamond\Psi$$

A safety property Ψ which states that *something bad must never happen* can be specified in terms of reachability as:

$$init :\rightarrow \forall\Box\neg\Psi.$$

and the main concern is to prove that the formula Ψ is never reached from the initial states.

With the help of HyTech, it is quite easy to prove this kind of property by checking that the two robots can be found in the shared area at the same time, i.e., no reachable state satisfies the *robot1* resides in location *SA1* and *robot2* resides in location *SA2* at the same time. Figure 6 shows how to check this property with HyTech.

```
1  automaton Robot1
2    synclabs:  request1, read1, send1,ack1;
3    initially  Init & y1=0 & k=0;
4   loc Init:
5     while True wait {}
6     when k=0  do {y1' = 0} goto move1;
7   loc move1:
8      while y1<=a wait {dy in [8/10,1]}
9      when True do {k'=1, y1'=0} goto move2;
10 loc move2:
11      while True wait {dy in [8/10,1]}
12      when y1 >= b & k=1  goto SA1;
13     when y1 >= b & k=0  goto Init;
14     when y1 >= b & k=2  goto Init;
15 loc SA1:
16      while True wait {dy in [8/10,1]}
17      when True do {k'=0, y1'=0} sync request1 goto read;
18  loc read:
19     while True wait {}
20     when True  do {y1' = 0} sync read1 goto Wait;
21 loc Wait:
22      while y1=< 4 wait {dy =1}
23      when y1 >= 4  sync request1 goto read;
24     when True  sync send1 goto Send;
25 loc Send:
26      while True wait {}
27      when True sync ack1 goto Init;
28 end

29 init_reach := reach forward from init endreach;
30 ext_error := loc[Robot1] = SA1  & loc[Robot2] = SA2;
31   if not empty(init_reach & ext_error)
32    then prints "Mutual Exclusion is violated";
33 else prints "Mutual Exclusion holds";
34 endif;
```

Fig. 6. Example from the HyTech code of the automaton Robot (Lines 1–28) Fig. 2. Some analysis commands are shown in lines 29–34.

The model checkers can check not only reachability of a certain property, but also time requirements between certain events or actions. For example, we can use HyTech to determine the maximum delay between signals sent to the robots. In particular, we can use HyTech to compute the value of the clock c in the set of all reachable states of the automaton *controller*.

HyTech additionally can be used not only to prove safety requirements, but also to find conditions on the parameters to guarantee the safety property. This is very useful to design parameters of robots specially when they are involved in critical situations. It should be noted that not all the model checking tools support such type of parametric analysis. One of such tools is HyTech [5]. For example, HyTech can compute the condition on the parameter **a** and **b** of the two robots to guarantee the existence of mutual exclusion.

5 Conclusion

In this paper we presented an approach that allows the specification of formal systems on different level of abstraction. We were interested in modeling and specification of safety critical surgical robots using hybrid automata. Particularly the area of surgical robots requires methods that guarantee the correctness of surgical system and its adherence to crucial safety properties. The paper main idea was to show how to coordinate the behavior of surgical robots under different levels of abstraction. Hence, we can take the approach of the paper to model a more complex safety critical scenario.

References

1. A. Hall, "Using formal methods to develop an atc information system," in *Industrial-Strength Formal Methods in Practice*, ser. Formal Approaches to Computing and Information Technology (FACIT), M. Hinchey and J. Bowen, Eds. Springer London, 1999, pp. 207–229.
2. A. Sobel and M. Clarkson, "Formal methods application: an empirical tale of software development," *Software Engineering, IEEE Transactions on*, vol. 28, no. 3, pp. 308–320, 2002.
3. T. Henzinger, "The theory of hybrid automata," in *Proceedings of the 11th Annual Symposium on Logic in Computer Science*. New Brunswick, NJ: IEEE Computer Society Press, 1996, pp. 278–292.
4. E. Clarke, O. Grumberg, and D. Peled, *Model checking*. Springer, 1999.
5. T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "Hytech: A model checker for hybrid systems," in *CAV '97: Proceedings of the 9th International Conference on Computer Aided Verification*. London, UK: Springer-Verlag, 1997, pp. 460–463.
6. G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," in *Hybrid Systems: Computation and Control, 8th International Workshop, Proceedings*, ser. LNCS 3414, M. Morari and L. Thiele, Eds. Springer, Berlin, Heidelberg, New York, 2005, pp. 258–273.
7. G. Behrmann, A. David, and K. G. Larsen, "A tutorial on Uppaal," in *Proceedings of 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems – Formal Methods for the Design of Real-Time Systems (SFM-RT)*, ser. LNCS 3185, M. Bernardo and F. Corradini, Eds. Springer, Berlin, Heidelberg, New York, 2004, pp. 200–236.
8. M. Capiluppi, L. Schreiter, J. Raczkowsky, and H. Woern, "Modeling and verification of a robotic surgical system using hybrid input/output automata," in *press European Control Conference, Zurich*, 2013.
9. R. Muradore, D. Bresolin, L. Geretti, P. Fiorini, and T. Villa, "Robotic surgery: Formal verification and plans," *Robotics Automation Magazine, IEEE*, vol. 18, no. 3, pp. 24 –32, sept. 2011.
10. Y. Kouskoulas, D. Renshaw, A. Platzer, and P. Kazanzides, "Certifying the safe design of a virtual fixture control algorithm for a surgical robot," in *Proceedings of the 16th international conference on Hybrid systems: computation and control*, ser. HSCC '13. New York, NY, USA: ACM, 2013, pp. 263–272.
11. L. Lamport, "A fast mutual exclusion algorithm," *ACM Transactions on Computer Systems (TOCS)*, vol. 5, no. 1, pp. 1–11, 1987.
12. S. Neema, "Analysis of matlab simulink and stateflow data model," Tech. Rep. ISIS 01-204, Vanderbilt University, Nashville, TN, Tech. Rep., 2001.
13. A. Mohammed and U. Furbach, "Mas: Qualitative and quantitative reasoning," in *Programming Multi-Agent Systems*, ser. Lecture Notes in Computer Science, L. Dennis, O. Boissier, and R. Bordini, Eds. Springer Berlin Heidelberg, 2012, vol. 7217, pp. 114–132.