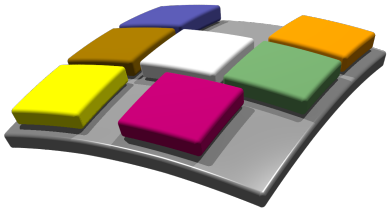


Cassting

Deliverable D3.4

Adapting Abstract Strategies to Real Platforms

Jiří Srba (AAU), Kim G. Larsen (AAU),
Marius Mikučionis (AAU), Petur Olsen (AAU),
Arne Skou (AAU), Erwin Fang (RWTH),
Nicolas Markey (CNRS)

Project	Cassting — Collective Adaptive Systems SynThesIs with Non-zero-sum Games		
Project id.	FP7-601148		
Workpackage	WP3	**Nature**	R (report)
Deliverable	D3.4	**Dissemination**	PU (public)
Date submitted	September 2014	**Date due**	September 2014
Version	1.0 (final)		

Adapting Abstract Strategies to Real Platforms

Jiří Srba Kim G. Larsen Marius Mikučionis
Petur Olsen Arne Skou Erwin Fang Nicolas Markey

September 2014

1 Introduction

The Cassting consortium have been active in this area already before the launch of the project, and the work described in this report partly builds on this background and demonstrates the recent development and future plans.

We shall first discuss the progress on the HomePort middle-ware platform that facilitates the communication among the different sensors and actuators in the smart house setting as outlined in Section 2. This platform is the bases for facilitating software-hardware interactions in Cassting case-studies as it provides a common API for the communication with hardware devices.

Another tool, Uppaal TRON, has been further developed in order to limit the communication latency between the adapter and the hardware platform and it was integrated with Uppaal TIGA in order to exploit the game framework on timed automata for the purpose of online and offline testing. Details are described in Section 3.

Next, we have built a proof-of-concept smart house model in order to demonstrate the possibility of running the whole tool chain in home automation on a cheap, credit-card size controller Raspberry Pi with limited resources. The successful case is described in Section 4 where we also outline the future plans regarding this case study.

Finally, we initiated a theoretical work on adapting the existing game theories so that they take into account the imperfection of communication with real hardware components and the synthesized strategies can compensate for this lack of precision, as explained in Section 5.

2 HomePort Platform for Smart Houses

Home automation is one of the key application areas of Cassting and here the HomePort middle-ware platform [GOR+13] developed at AAU provides the interface between the software and hardware components in a smart house, facilitating an easier communication with the sensors and actuators present in a house.

2.1 Executing Home Automation Controllers

A diversity of services can be provided and supported by home automation (or Ambient Intelligence (AmI)) systems. They can control heating systems, gather temperature or humidity data, manage lighting, or manipulate household appliances. The goal of home automation is to coordinate and control these different services in order to create an intelligent environment. The devices are often made available through a diversity of networks, protocols and technologies which makes it impossible to access the devices in a common manner.

Designing and setting up an Ambient Intelligence (AmI) system is a tedious task because of the variety of networks. One may think that this is a once only problem when a system is installed. Nevertheless, when a building is planned, its AmI technologies are chosen based on the current availability, performance and price; but after some years a building administrator may wish to update or renew the system, and by then the technologies that were chosen are no longer the best ones, are outdated, or no longer supported. If the design and implementation then rely heavily on the technological choices, it might be impossible to change the system, and therefore it cannot be updated. However, if the design has a clear interface to the heterogeneous networks from the beginning, technology changes will not affect the main system, and updating or changing a component is localized. Thus, it is justified to invest in a middle-ware with a service oriented interface to technologies.

Recently systems have been developed for aggregating access to devices of different vendors under a single system. One such system is the open-source system HomePort. HomePort provides homogeneous access to heterogeneous subnetworks through a RESTful interface. Using such a system not only makes it possible to combine devices for different vendors, it also separates the controller implementation from the specifics of the devices. This allows for interchangeable controllers and controllers implemented in different languages or frameworks. Allowing the controller to access devices through a RESTful interface gives access to a wide variety of networks, protocols, and vendors.

We exemplify this approach through a controller developed in the model-checking tool Uppaal interfacing with devices through HomePort. Other systems could be used in a similar manner. E.g. Seluxit, an industrial partner of the Cassting project, develops control boxes which provide homogeneous access to devices on different networks. Controllers can be developed using different tools, e.g. Simulink, Labview, or SCADE.

2.2 The Uppaal-HomePort Toolchain

This section provides an overview of the toolchain and its recent development, starting with a description of the two existing components, HomePort and Uppaal. This is followed by a description of the two novel components, HomePort2Uppaal and the UppaalInterpreter.

HomePort is a middle-ware implementing a common interface to heterogeneous home automation networks. It facilitates the tasks of the control system

by enabling it to access and modify the services of the environment in a unified manner, regardless of the protocol or technology that a specific device is using. Moreover, its REpresentational State Transfer (REST) architecture is easy to interface with. It also provides an event mechanism allowing the control system to receive notifications when the state of a service changes. HomePort plays two roles in the toolchain: it provides a list of available services and enables read and write access to these services.

Uppaal is a toolbox for modelling, simulation and verification of real-time systems. It is composed of three main parts: a description language, a simulator, and a model checker. The description language models system behavior as communicating state machines with timers. The simulator enables exploration of possible executions of a model. The model checker can check safety, liveness and reachability properties on a model. In the toolchain, Uppaal is first used to model the controller using the description language through a graphical user interface. The model checker is then used to verify properties of the designed model. Finally, the experimental concrete simulator is used to assist in executing the control system. The concrete simulator, as opposed to the symbolic simulator, uses concrete time values rather than symbolic intervals.

The purpose of *HomePort2Uppaal* is to translate a list of HomePort services to a Uppaal template that enables controller models to access the services. HomePort2Uppaal establishes communication with a HomePort server and retrieves its service list. From this list it generates a Uppaal template containing communication channels and global variables that enable controller models to access and modify the state of services, as well as receiving events. To exchange data between this template and others, one-way unconditional synchronous value passing is used.

The *UppaalInterpreter* runs an execution of the control system on the real hardware platform. The model is first given to a Uppaal simulator. The UppaalInterpreter then executes it by telling the simulator which transitions to take, by running the algorithm as described below. The first step of the algorithm is to retrieve all active transitions from the simulator. It then takes the first active internal transition that can be taken at the current time. An internal transition is a transition that does not involve synchronization with the HomePort process. After taking any transition, the algorithm is restarted, as the set of available transitions is no longer valid. Once all internal transitions that do not involve time passing have been taken, it checks if an event was received from the HomePort server. In this case, the transition corresponding to the received event and its value is taken. When all events are processed, it takes the first transition involving communication with HomePort. When taking such a transition, the corresponding request is performed on the HomePort server. Once there are no more transitions that can be taken at the current time, it waits for the minimum time until a transition can be taken again, or until an event is received. The UppaalInterpreter is responsible for synchronizing Uppaal time with external time. The UppaalInterpreter currently uses seconds as a unit, but this could easily be modified to adapt to different time scales.

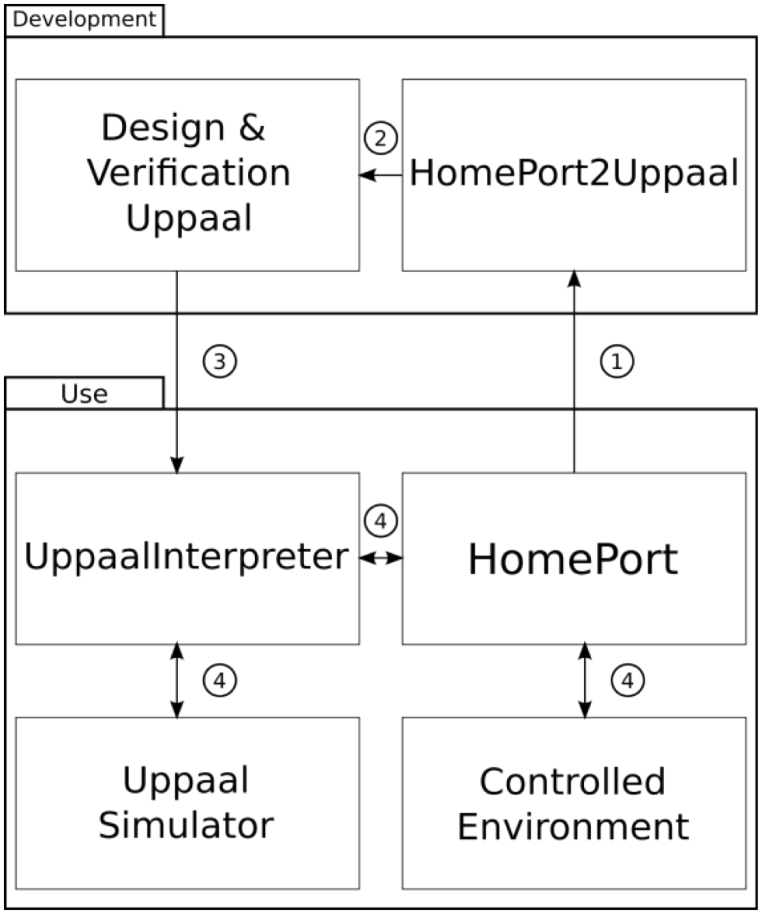


Figure 1: HomePort toolchain overview

2.3 Development and Use of HomePort

Figure 1 shows the different components of the toolchain and the different steps to go through from the development to the execution of a control system. The first step is to retrieve the set of services that are available in the environment, corresponding to the arrow labeled 1 in the figure. This is done by the HomePort2Uppaal module making a request to the HomePort web server, which returns an XML file with a description of all the services and information on how to access and use them. HomePort2Uppaal then transforms this list of services into a Uppaal template providing communication channels and variables for other templates to be able to access and modify the service states. This template can then be opened in Uppaal, represented by the arrow labeled 2. The actual control system can then be modelled as one or more Uppaal state machines. The verification and simulation features of Uppaal serve to

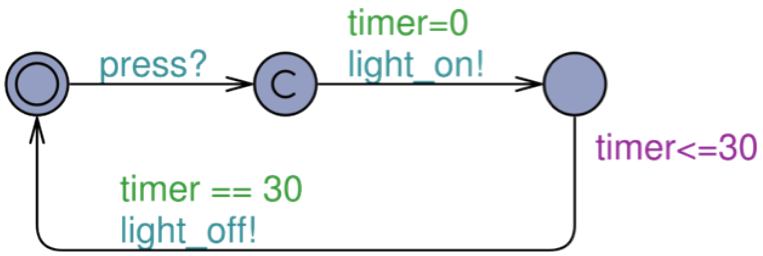


Figure 2: Example controller

check properties of the controller. Once the model of the controller has been constructed and verified, it is exported as a standard UPPAAL XML file.

The process can now move to the use phase, represented in the figure by the arrows labeled 4. The generated XML file is given to the UppaalInterpreter, which will execute the modelled controller, using a UPPAAL simulator. The execution is done by choosing appropriate transitions in the model, and by communicating with the HomePort server to get, set or receive events on the services. It also synchronizes the UPPAAL time with real world time in order to have a faithful real-time execution.

As an example consider the controller in Figure 2. The controller waits for the user to press the button. It then turns on a light and uses a timer to wait 30 seconds before turning the light off. Using the algorithm described above, this controller would wait for the "press?" event to be received. It would then perform the action to turn the light on through HomePort and take the "light_on!" transition. Then it would wait 30 seconds before the next action is available. Finally it would turn the light off through HomePort, take the "light_off!" transition, and return to waiting for an input.

2.4 Summary

In the beginning of the Cassting project we invested a considerable effort to complete the HomePort tool chain so that it provides a reliable middle-ware platform for the interaction with the real hardware in the home automation setting. As documented by [DGM+13], this effort succeeded and we can now provide a fully featured interface for executing the different strategies investigated by the partners in the project. HomePort system is being continuously developed and optimized and in the remaining deliverables of the Cassting project we will evaluate the applicability of the approach on larger case studies. For this purpose, the AAU team in cooperation with Seluxit is at the moment working on building a scaled model of a smart house, including the advanced termodynamics of the floor-heating, so that it becomes in the near future available for experimental evaluation.

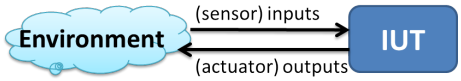
References: [DGM+13]

3 Online Testing of Real-Time Systems

We shall now describe a branch of the tool UPPAAL, called UPPAAL TRON, that allows for a direct communication with a real hardware platform in order to execute an interactive communication for the purpose of online testing. This allows us to relate abstract models to concrete physical processes in the tested system.

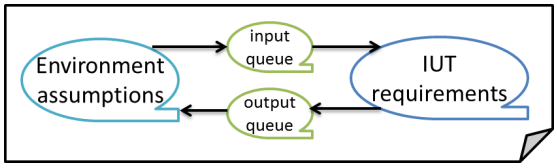
3.1 UPPAAL TRON

UPPAAL TRON is an online testing tool for real-time black-box systems where the internal state of the implementation under test (IUT) is not accessible. The tool tests for observable timed input-output conformance against UPPAAL timed automata model, where the inputs are controlled by the tester and outputs are controlled by the IUT. The test setup (see Fig. 3a and 3b) assumes that the testing tool plays the role of the IUT's environment, communicates with the IUT and decides whether the timed input-output behavior exhibited by IUT conforms to the model. Informally, IUT conformance to a particular model means that the observed behavior of IUT is included in the behavior of the model: all timed outputs can be matched in the model and any extra output or wrong timing (too early, too late or no show) is considered as non-conformance witness. Thus timed input and output event sequences is the only interface relating the model with physical system.



(a) Embedded system deployment

(b) UPPAAL TRON test setup



(c) UPPAAL model setup

Figure 3: UPPAAL TRON testing framework

Figure 3c shows that model mimics the test setup and consists of two parts: assumptions about the environment (used for generating timed input stimuli) and requirements for the IUT (used for checking the correctness of the timed output responses). Both the environment and the IUT models are formulated as

a set of communicating timed automata processes and the communication between them is regarded as observable input and output events. The online test is performed by emulating the environment model, generating input events, executing corresponding input stimuli, recognizing an output responses and checking the corresponding output events against possible responses described in the IUT model. The inputs and outputs are marked by channel synchronization events in the model. Uppaal TRON also provides a possibility of attaching model variable values to inputs and checking the outputs values with model variable values.

The communication between tester and IUT is assumed to be asynchronous: once the input is sent the IUT cannot refuse (IUT is input enabled) and the tester must consume any output produced by IUT likewise. The translation between abstract model events and physical events is done by a test adapter software. Uppaal TRON provides a number of sample adapter implementations using C-library interface (dynamic library function calls), TCP/IP sockets (streams of binary data) and standard input/output (textual trace). In practice, the adapter software (and hardware) inevitably introduce communication latency between tester and IUT, moreover there is also a possibility of input and output interleaving inside adapter and therefore the adapter itself is modeled as part of requirements for IUT.

3.2 Online Testing

Uppaal TRON employs online testing paradigm by deriving, executing and monitoring the outcomes at the same time, and thus can cope with large degree of non-determinism within the model which allows to have many parallel (interleaving) processes and also flexibility in event timing. This is not the case in offline test generation where every possible outcome needs to be predicted ahead of test execution, which in turn produce exponentially large test cases. Some of this non-determinism happens in the adapter layer and thus unavoidably leads to loss of precision when monitoring the correctness of the real-time IUT.

Our recent work [DLM+13] addresses a special case of generating test cases for real-time test adapters where the communication latency is limited. Figure 4 shows a simple test sequence $\Delta a \delta b \Delta$ to be executed at IUT side and a problem is to derive $a' \Delta \delta \Delta b'$ sequence at the tester's side which triggers the required sequence at IUT side. The paper introduces a Δ-testability criterion and shows that the conditions for the correctness monitoring during remote testing can be as precise as in local testing setup as if without adapter latency. The main idea is that if individual input and output events are separated by at least $2 \cdot \Delta$ time duration where Δ is the maximum delay of one-way communication, then we can avoid input and output signal interleaving during communication between the tester and the IUT, and hence we can reliably reconstruct the order of those events, which is crucial for monitoring the correctness and stimulating the right input sequence. Moreover, with Δ-testability property we can transform a local timed sequence (to be executed at IUT side) into a remote timed input sequence (to be executed at the tester side) and hence generate a remote offline test case.

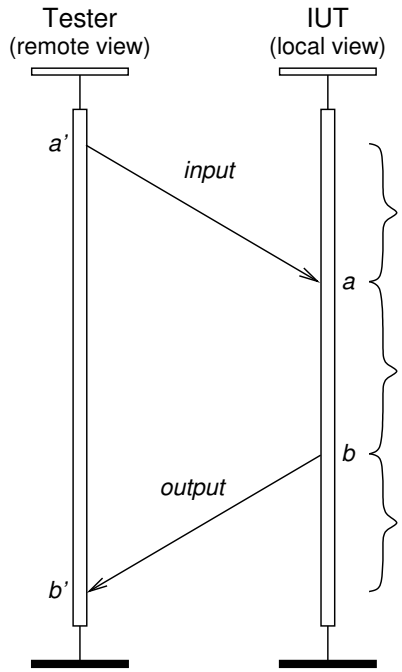


Figure 4: Message sequence chart of remote test execution

The second contribution is a practical test framework. The idea of the method is to use Uppaal TIGA to decide whether it is possible to generate a test sequence suitable for offline testing given the adapter assumptions and IUT requirements into account. If a sequential test is not possible, then the model can be changed or Uppaal TIGA can be used to generate a strategy against the uncontrollable adapter which can be executed as a remote test case against IUT. The benefit of test sequence is that it is simple and predictable in offline execution and regression testing while a strategy generation is adaptive and hence more powerful but may require exponential number of memory resources (in terms of simultaneous input and output signals in the adapter). Figure 5 shows the model setup accepted by Uppaal TIGA which can then synthesize a remote test sequence (or an adaptive strategy) when given a test purpose (e.g. forcing IUT into some state) against uncertainties inside adapters. In this particular example we use a typical case with asynchronous FIFO queues modeling input and output signals, but in principle Uppaal TIGA accepts any adapter model.

3.3 Summary

The recent work on Uppaal TRON focuses on limiting the communication latency between the adapter and real hardware platform in order to improve

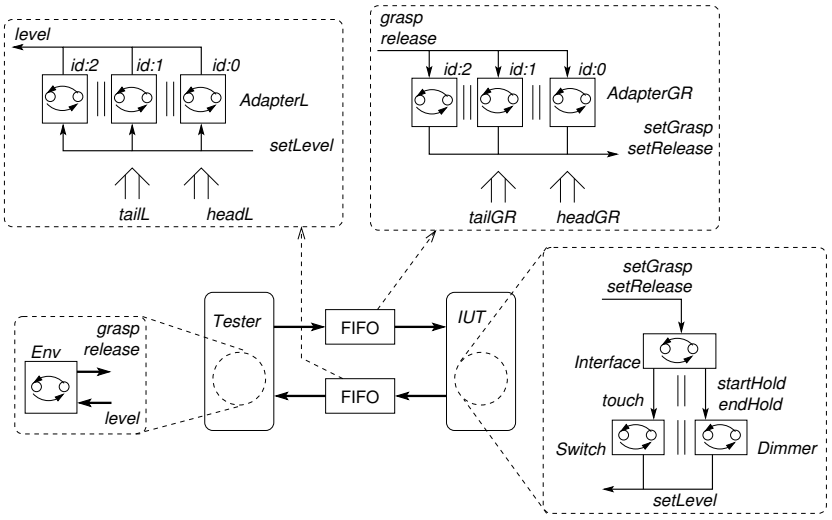


Figure 5: Model setup for remote testing

the precision and reliability of online testing and on the integration with UPPAAL TIGA in order to reuse the game framework on timed automata to facilitate testing, using the fact that testing can be considered as a game between an uncontrollable adapter and the controller. The tool UPPAAL TRON has been equipped with these new possibilities and creates a link between the abstract models and control strategies and their execution counterparts running on a concrete hardware platform.

References: [DLM+13]

4 Proof-of-Concept Smart House Demonstrator

In order to further explore the possible interaction and execution of game strategies on different hardware platforms, we proposed a master thesis project [Sør14] that should demonstrate how abstract strategies can be executed in a smart house setting using a cheap embedded controlled based on the Raspberry Pi open-source platform. This project contributed both to the theory of game synthesis as well as to the adaptation of the synthesized strategies to real platforms, so it is reported also in Deliverable D4.2. Here we focus solely on the features related to the hardware platform and communication with the smart house.

4.1 Home Automation Framework

Home automation systems consist of a large variety of devices, each providing one or more services. Some of these services provide sensor inputs (e.g. switch

clicks, temperature readings, light level readings, movement detection, window state) and are said to be *uncontrollable*, as they only output an observation of the system. Other services also control the state of some physical device (e.g. thermostats, lamps) and are said to be *controllable* as the state of these can be observed, but also changed (programmatically). A device may provide both controllable and uncontrollable services, e.g. a thermostat with a built-in temperature sensor. Figure 6 shows an example of two uncontrollable services (a switch, top left and a thermometer, bottom) and a controllable service (a lamp, top right) and their state-space.

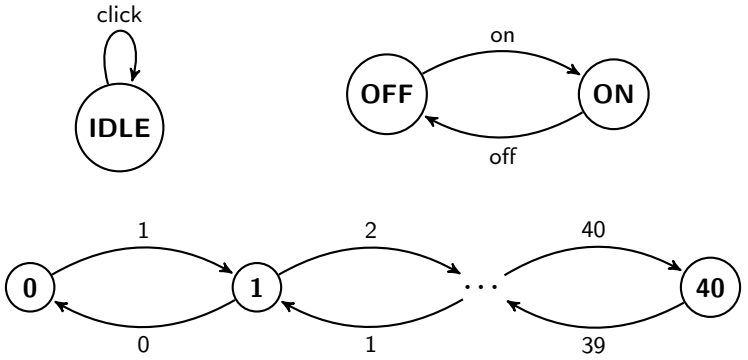


Figure 6: Example of two uncontrollable services (a switch, top left and a thermometer, bottom) and a controllable service (a lamp, top right) and their state-space.

Home automation allows these services to interact with each other in order to exchange information. This information can be used to make decisions on behavior based on the global state of the system and is used for devices to operate in an optimal manner (e.g. minimizing energy consumption). An example of this is a system which turns off the heat when a window is open. This requires an exchange of information between the window sensor and the thermostat.

Figure 7 shows an example of a home automation setup in a hallway. In this scenario, we have seven services: three lamps (controllable), three switches (uncontrollable) and a motion sensor (uncontrollable).

The aim in home automation is for the system of various services to interact and cooperate to meet some desired control objective. Figure 8 provides an overview of the general setup in home automation. A controller has access to information about controllable and uncontrollable services and may use this information to dictate behavior of controllable services (in order to meet the control objective). Dually, the environment is affected by the state of controllable and uncontrollable services and dictates behavior of uncontrollable services (this covers any change to the system that is not controlled by the controller, e.g. user input or physical effects on service states).

A general setup for a system uses the HomePort platform described in Section 2 with a centralized controller component. While system control could be

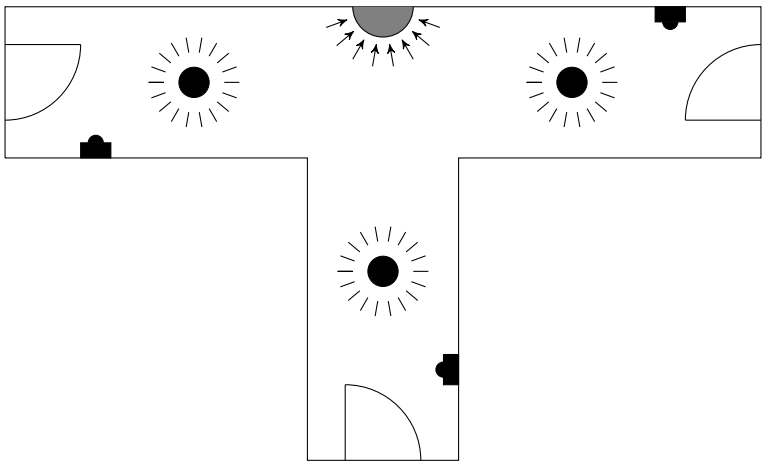


Figure 7: Home automation scenario. Black boxes are switches, black circles are lamps and the gray semi-circle is a motion sensor.

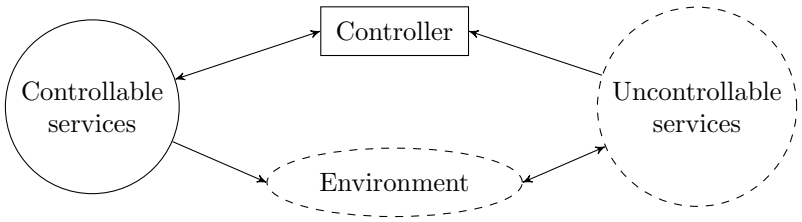


Figure 8: General setup in home automation

handled in a distributed manner, due to the necessity of middle-ware software for service interaction, as well as limited computational power and high demands for a minimal power consumption of devices, a centralized controller component is commonly desirable.

4.2 Hardware Representation

HomePort represents service states as an integer value from a predefined interval, e.g. a lamp has the state interval $[0, 1]$ where the state 0 is interpreted as *off* and the state 1 is interpreted as *on* and a switch has the state interval $[0, 0]$ where the state 0 is interpreted as *idle*. In other services, e.g. a thermometer, the interpreted valuation is equivalent to the integer value and is referred to as a *reading*. We denote the non-negative integer valuation of a state s as $\nu(s)$. State valuations define a complete ordering of states by the relation of integers, e.g. for a lamp where $\nu(\mathit{off}) = 0, \nu(\mathit{on}) = 1$, then $\mathit{off} < \mathit{on}$.

Finally, we make some assumption about the behavior of home automation systems:

1. **Instant communication.** We assume communication in a home automation system to be instant, i.e. it takes no time to change or obtain the state of a service. This assumptions is justified by network communication being practically instant and thus that any delay will be negligible.

2. **Non-zero delay between state changes.** We assume that a non-zero delay should always occur between changing the state of a controllable service. This assumption reflects that no physical operation is in fact completely instant in a continuous timed domain.

3. **Non-Zeno behavior.** For similar reasons as above, we assume that the system will not by itself show Zeno behavior, i.e. will not change state infinitely often in a finite amount of time.

These assumptions should be reflected also in a control strategy, that is, should not be violated under any control strategy.

4.3 Hardware Platform and Execution Details

We now describe a proof-of-concept implementation of a completely automated controller synthesis toolchain. Figure 9 shows the toolchain of the implementation. The entire toolchain—including strategy synthesis by Uppaal TIGA—is executed on a Raspberry Pi (Model B), a cheap (35€), low-powered credit-card sized computer featuring a 700 MHz ARMv6 processor and 512 MB of RAM equipped with a WiFi dongle for wireless interaction with the system.

Interaction with the system happens through a simple web-interface in form of a *web app*. The app automatically detects changes to the underlying home automation system and allows easy configuration and management of devices.

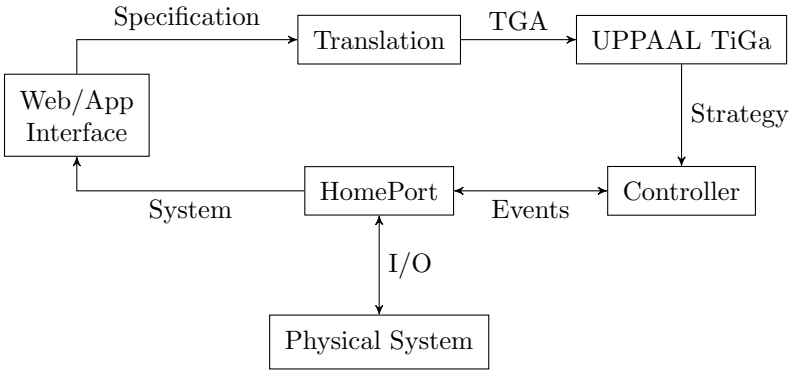


Figure 9: Toolchain for automated controller synthesis

A rule editor facilitates easy point-and-click management of behavior specifications. Given a specification, a control strategy is automatically synthesized and a corresponding controller is constructed and invoked.

Based on the system description and specification provided from the interface, translation to Uppaal TIGA models is performed in PHP. The model is automatically provided to Uppaal TIGA, which has been compiled for the ARMv6 processor and is executed directly on the Raspberry Pi. The synthesized strategy is used to dictate behavior using the controller.

The controller is a small control program, written in `C++` for portability. Provided with a system description, a specification and a strategy, the controller monitors the underlying HomePort system and effectuates behavior according to the strategy, effectively providing an implemented control program.

4.4 Summary

The thesis of Mathias G. Sørensen clearly demonstrated the feasibility of the proposed approach and he succeeded in executing the whole tool chain on the hardware with very limited resources. His thesis was one of two that received the Embedded Award 2014 by Danish Industry for the best Danish master thesis on embedded systems. We are currently extending the proof-of-concept into a scaled-down model of a real house with 25 switches and 18 lights, equipped with further sound and motion sensors, in order to explore the scalability of the proposed solution.

References: [Sør14]

5 Permissive Multi-Strategies in Timed Games

Strategies, as we define and use them in our formal approached, are not directly suited to the execution on real systems. In particular, we see them as a

mapping from any given history to a move for the considered player, without caring about how such a mathematical object can be implemented on a real platform. In the setting of real-time systems, abstract strategies are assumed to have arbitrary precision (the proposed move will be played at the exact date), while implementations cannot achieve such a precision. We propose to adapt the notion of strategies in real-time games in order to address (a part of) this discrepancy.

5.1 Permissive Abstract Strategies

When considering games for synthesis, it is usual to see strategies as functions returning, for each history of the play, one possible move (i.e. an action) to be played by the considered player. This definition is also used when considering *timed games*, meaning games played on timed automata: in that setting, a move consists of an action and a delay. Once all the players have proposed such a move, the player with the shortest delay is selected, the chosen delay is elapsed, and the proposed action is played.

What is problematic with such a definition is that a strategy can, for instance, propose convergent delays in order to "block" time elapsing and prevent some other player from playing her action. Such convergence phenomena, known as Zeno behaviours, have no physical counter-part, and our aim is to forbid strategies that rely on such phenomena that are not executable on real hardware platforms. For this, we adapt the notion of strategies: our *permissive strategies* now return an action and an *interval of delays*. The exact delay will be chosen (by the opponent in the game) within the proposed interval, in order to match the worst case. In order to favour large intervals of delays, we associate with each interval a penalty, with the intention to have larger penalty for smaller intervals. Our aim is then to find optimal winning strategies, i.e. strategies that achieve their goal and propose intervals as large as possible.

5.2 Proposed Solution

The setting presented above is very large, and many questions remain open. We solved one precise case in [Fan14] where:

- the game is two-player, zero-sum, with one player having a reachability objective,
- there is one clock in the timed game,
- the penalty associated with an interval $[a, b]$ with $a < b$ is $1/(b - a)$ while punctual intervals have penalty $+\infty$, and
- penalties are summed up along the outcomes.

For this setting, we proposed an algorithm to compute the optimal penalty that the protagonist can achieve, and in particular, to decide if there is a winning strategy avoiding punctual intervals. We proved that optimally-permissive

strategies can be chosen memoryless (which is the key property to ensure that the algorithm terminates and it is correct), and we showed how we can compute such strategies.

This preliminary work opens large avenues for future work. First of all, we are currently implementing our algorithms in order to evaluate how they perform in practice. As immediate continuations to the work reported here, we would like to precisely characterize the complexity of the problem, and to make sure that our algorithm is optimal in terms of complexity. We believe that extending the problem to timed games with several clocks would lead to undecidability, but this remains to be proven. Adapting this approach to more complex objectives is also a natural and desirable extension.

More challenging future developments include taking imprecisions into account also in the observation of the history: currently, strategies have perfect knowledge and precision about what has happened in the history, which obviously is too precise and unrealistic. Defining some kind of granularity of observations, and again trying to maximize the required granularity in order to achieve a given objective, would certainly make our approach even more relevant for timed games.

5.3 Summary

The work on permissive abstract strategies a first step towards a general game theory that is aware of the imperfect communication between software and hardware components and explicitly models the inaccuracy in timing of both controllable and uncontrollable events. It is a necessary step on the transition from mathematically sound control strategies into practically executable strategies on a given hardware platform that should guarantee correct behaviour even in the case where the timing is imprecise. The work described in this section was done in cooperation between Aachen and CNRS where a master student Erwin Fang from Aachen visited Paris for one-year Erasmus exchange program.

References: [Fan14]

6 Conclusion

The progress described in this deliverable follows the expected plan and demonstrates the commitment to push the theoretical results achieved by the members of the consortium into real executable platforms. Most of the contributions in the deliverable has been done by the AAU team due to the fact that they are responsible for the tool development based on UPPAAL tool suite and because they have the tightest links with the Danish industrial partners. In the second half of the project, we expect that the other partners will exploit the frameworks that have become available and use their techniques to synthesize controllers that can be then directly executed on the given hardware platform. The progress described in this deliverable is regularly communicated to the partners of the

consortium at the meetings and workshops. Most of the work in this deliverable focuses on implementation of the concrete communication with hardware components and has therefore less visibility in terms of the number of publications. This was expected and the investment will return in the second part of the project once the developed software solutions will be exploited by the partners of the Cassting consortium.

List of Cassting Publications

[DGM+13] Peter H. Dalsgaard, Thibaut Le Guilly, Daniel Middelhede, Petur Olsen, Thomas Pedersen, Anders P. Ravn, and Arne Skou. A toolchain for home automation controller development. In Onur Demirörs and Oktay Türetken, editors, *39th Euromicro Conference on Software Engineering and Advanced Applications(SEAA'13)*, pages 122–129. IEEE, 2013. Available from: `http://dx.doi.org/10.1109/SEAA.2013.36`, `doi:10.1109/SEAA.2013.36`.

[DLM+13] Alexandre David, Kim G. Larsen, Marius Mikucionis, Omer Nguena-Timo, and Antoine Rollet. Remote testing of timed specifications. In Hüsnü Yenigün, Cemal Yilmaz, and Andreas Ulrich, editors, *Testing Software and Systems - 25th IFIP WG 6.1 International Conference, ICTSS 2013, Istanbul, Turkey, November 13-15, 2013, Proceedings*, volume 8254 of *Lecture Notes in Computer Science*, pages 65–81. Springer, 2013. Available from: `http://dx.doi.org/10.1007/978-3-642-41707-8_5`, `doi:10.1007/978-3-642-41707-8_5`.

[Fan14] Erwin Fang. Permissive multi-strategies in timed games. Research Report LSV-14-04, Laboratoire Spécification et Vérification, ENS Cachan, France, March 2014. 36 pages. Available from: `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2014-04.pdf`.

[Sør14] Mathias G. Sørensen. Controller synthesis for home automation. Master's thesis, Aalborg University, Department of Computer Science, Denmark, 2014.

References

[GOR+13] Thibaut Le Guilly, Petur Olsen, Anders P. Ravn, Jesper Brix Rosenkilde, and Arne Skou. Homeport: Middleware for heterogeneous home automation networks. In *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM'13)*, pages 627–633. IEEE, 2013. Available from: `http://dx.doi.org/10.1109/PerComW.2013.6529570`, `doi:10.1109/PerComW.2013.6529570`.