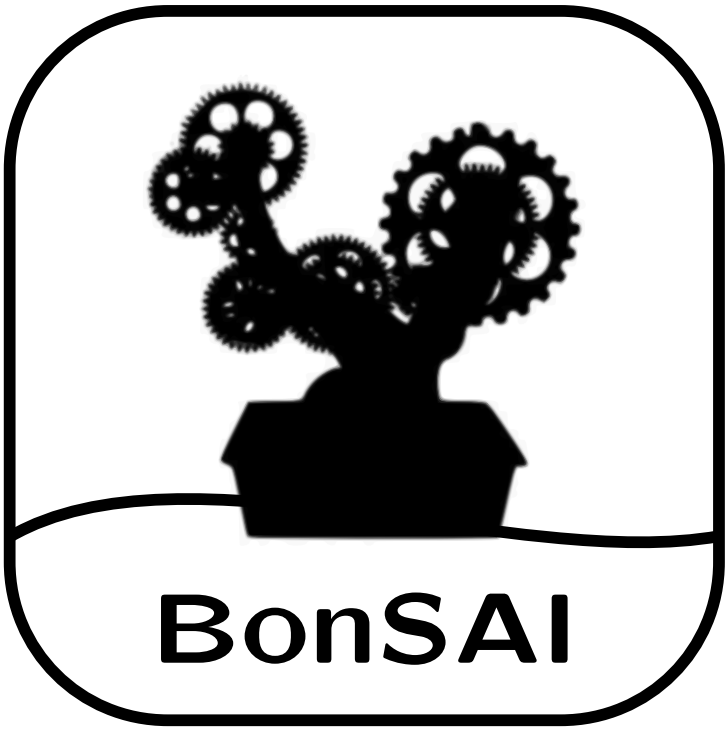
Behavior Coordination for Reusable System Design in Interactive Robotics



Frederic H. K. Siepmann

Behavior Coordination for Reusable System Design in Interactive Robotics

Der Technischen Fakultät der Universität Bielefeld
zur Erlangung des Grades
Doktor-Ingenieur
vorgelegt von
Frederic H. K. Siepmann
Bielefeld - 12. April 2013

Gutachter:
Dr.-Ing. habil. Sven Wachsmuth
Dr.-Ing. Marc Hanheide, Lincoln Centre for Autonomous Systems, University of Lincoln

Prüfungsausschuss:
Prof. Dr.-Ing. Ulrich Rückert
Dr. Jonathan Maycock
Verteidigt und genehmigt am 29. Juli 2013.

Gedruckt auf alterungsbeständigem Papier nach ISO 9706.

Abstract

Robotic systems research is typically a result of a collaborative engineering process in an environment of rapidly changing technologies. It involves a large number of hardware and software components, each of which are problem solutions for different challenges from different research areas. Systems for Human-Robot Interaction face the additional challenge of having to actually work together with humans in a shared environment.

Despite the fact that some single capabilities needed to solve various tasks with robotic systems are well established, however, robots frequently fail when they need to combine these capabilities and demonstrate them in a complex real world scenario. The aspect of coordinating and efficiently combining robots capabilities is one area where robots still fail.

The focus of this work is to provide a framework for developers of interactive robot systems that perform in domestic environments, which allows the combination and improvement of building blocks of the robot behavior based on experience gained in real world interaction and make the combination and coordination of these building-blocks easier and more easily reusable for developers.

With the framework developed during this thesis there where many occasions where a robot platform was evaluated in a real world environment. This iterative design process documented here helps to answer questions about how to improve the robot performance based on observations from real world interactions, how to enable re-usability of robot behavior building blocks across scenarios and platforms and how to combine and coordinate the different robot capabilities from a developers point of view.

Acknowledgments

First of all I would like to thank Dr. Sven Wachsmuth and Dr. Marc Hanheide for agreeing to supervise my work and for their valuable input throughout the process of this thesis. You always took your time for discussions and advice that paved the way not only for me but also for my robot and a whole RoboCup team from Bielefeld. Thank you!

Speaking of the RoboCup team I would like to thank all ToBIs out there for their hard work and commitment during the various competitions. It has been a great time not only meeting all of you but also working with you - and supervising at least some of you. Thank you all, it has been a great experience and a great success!

Working in Bielefeld was a wonderful experience for me with colleagues from different areas of research. There have been inspiring discussions with many of you. I do hope this spirit remains in Bielefeld. I do want to thank a few people directly that really helped me by sharing their knowledge and with their willingness to discuss: Dr. Ingo Lütkebohle is up front. I have never met a person that shares and discusses his deep knowledge so passionately, thank you!

I would also like to thank Florian Lier, who is not only a diligent and knowledgeable colleague but also a true friend. Thank you Fl0.

I want to thank Simon Schulz, who shared his office with me during my thesis. Simon is not only an exceptionally gifted (and creative) hardware hacker, he is also a robot rescuer, an exemplary dad, climber and friend. Thx Simon!

Leon Ziegler, who accompanied me during the RoboCup endeavor from the very beginning, started as a ToBI and has become a great colleague and friend. Thank you Leon for many discussions and even more hacking sessions.

I want to thank Manja Lohse for her collaboration and her commitment to use my robot in the real world. Thank you for your endurance and patience, not only with me but also with BIRON.

The list of people who I would like to thank is too long too mention all of them here, but I want to add my students that helped to make BonSAI a success: Johannes Wienke, Torben Töniges, Norman Köster and Lukas Kettenbach. Thank you guys for hacking, testing and crashing BonSAI.

Contents

1. Developing Interactive Robots - A First Contact **1**
- 1.1. From Service to Personal Robots . . . 3
- 1.2. Research Questions . . . 5
- 1.3. Contribution & Outline . . . 6

2. Interactive Robots: Software & Systems **9**
- 2.1. Scenarios for Interactive Systems . . . 9
  - 2.1.1. Coordination: Arbitration and Command Fusion . . . 12
  - 2.1.2. Control Principles: Deliberative, Reactive or Hybrid . . . 13
- 2.2. Robot Architectures in the wild . . . 17
  - 2.2.1. Roblet®Technology . . . 18
  - 2.2.2. Graphical Tools: Choreographe and NAOqi . . . 20
  - 2.2.3. The Behavior Markup Language (BML) . . . 21
  - 2.2.4. The Robot Operating System (ROS) . . . 23
- 2.3. Summary . . . 26

3. The Tool Set: Platform and Communication Framework **27**
- 3.1. Bielefeld Robot Companion (BIRON) . . . 27
- 3.2. BIRON Hardware . . . 30
- 3.3. An Active Memory for Information-driven Integration . . . 32

4. The System Foundation & Concepts **33**
- 4.1. Requirements for Interactive Frameworks . . . 36
- 4.2. Bonsai Architecture: Design Principles . . . 39
  - 4.2.1. Separation of Concerns (SoC) & Modular Programming . . . 39
  - 4.2.2. Abstraction & Readability . . . 39
  - 4.2.3. Loose Coupling & Information Passing . . . 40
  - 4.2.4. Sensors: Interface for acquiring information . . . 42
  - 4.2.5. Actuators: Interface for acting . . . 42
  - 4.2.6. Skills: Behavior Modules Facilitating Strategies . . . 43
- 4.3. Contribution . . . 48

5. Implementing the Bonsai Framework **51**
5.1. Communication: Abstraction for Consistency 52
5.1.1. Sensors and Actuators 53
5.1.2. Data Content: Necessity for decoupling 55
5.2. Control Flow: Enabling Reusable Building-blocks 57
5.2.1. Skills 58
5.2.2. Control Layer and SCXML 61
5.3. Contribution 64

6. Evaluation **67**
6.1. System Evaluation: The Extended Home Tour 67
6.2. Performance & Stability: The RoboCup Experience 72
6.3. Portability & Usability: Bonsai Developer Studies 79
6.3.1. Bonsai Portability 80
6.3.2. Bonsai Usability 82

7. Discussion & Conclusion **89**
7.1. Interactive System Design with Bonsai 89
7.2. Iterative Behavior Modeling: From User Studies to System Design . 92
7.3. Future Work 94

A. Bonsai Implementation **97**
A.1. Bonsai Configuration XML Schema 97
A.2. Bonsai Configuration File Example 99
A.3. Bonsai SCXML Finals 2012 109

B. Bonsai Evaluation **113**
B.1. Team ToBI Programming Experience in 2009/2010 114
B.2. Usability Study Instruction Sheet 114
B.3. Additional Help Sheet 117

List of Figures

1.1. **left:** Different robot applications and resulting requirements for robotic systems. **right:** The robot Cosero of the RoboCup team NimbRo doing the registration task at RoboCup 2011, Istanbul. . . 4

2.1. Example from a Discrete Event System (DES), showing a *traverse door* state. 12
2.2. Deliberative System according to *Sense-Plan-Act* and *Sense-Plan-Model-Act* (dashed). 14
2.3. Diagram of a *Subsumption* architecture with 4 layers of competence. 15
2.4. The software architecture of the robot TASER where Roblet-servers (RS) are used to provide a hardware abstraction layer, taken from [6]. 18
2.5. Example of two applications (app1/app2) running distributed on the NAO robot and on computer1. Applications can use all functions registered to the main broker (local/remote call). 19
2.6. A screenshot of the Choreographe User Interface. 20
2.7. ROS smach 24
2.8. Example state machine with smach *outcomes*. 25

3.2. BIRON software architecture with central *Execution Supervison (ESV)* 27
3.1. **left:** The 2007 BIRON system waiting in the living room of the real world apartment of the robot. **right:** Schematic view of the BIRON system with its components. 28
3.3. Updated BIRON software architecture with *active memory* enabled. 28
3.4. **left:** The 2010 BIRON system waiting in the living room of the real world apartment. **right:** The 2011 BIRON system and its components on the right. 29
3.5. The evolution of the BIRON system from approx. 2002 (left) until 2012 (right). 30
3.6. Conceptual drawing of **component A** and **component B** communicating via an *active memory*. 32

4.1. Schema of the *Bonsai Architecture* style in more detail: Each layer processes information generated in layers underneath or in the same layer, enabling a semantic decomposition of the components. Only selected components of the *functional components* layer of the BIRON system (see Sec. 3.1) are shown. 34
4.2. Schema of an *Bonsai Modelling Concept* in more detail: 41
4.3. Schema of the implementation of the *person tracking* of the BIRON system from 2009-2011. 43
4.4. Schema of the implementation of the *navigation* of the BIRON system of 2009-2010. 45
4.5. Schema of an *informed strategy* in more detail: The *sensor fusion* on the left generates information for the *strategy*. The *Actuation* generates e.g. a goal to which the robot can navigate. 46

5.1. Schematic view of the use and implementation of *sensors/actuators* in Bonsai with the *NavigationActuator* as example. 55
5.2. Simple state diagram. 59
5.3. A simple statechart. 59
5.4. The life cycle of a Bonsai *skill*. 60

6.1. The robot apartment. The path of the tour (green) starts in the living room via the hall to the dining room. 68
6.2. The programming experience of the team ToBI members in 2012. An accumulated graphic for 2009/2012 as well as for 2011 can be found in B.1 73
6.3. Availability of the Bonsai features through the different years of the RoboCup competition. 74
6.4. Assessment of the RoboCup participants 2012 of the additional Bonsai modeling concept. 77
6.5. Practical Usage Assessment of Bonsai during the RoboCup competition from 2009/10-2012. The according questions (**bold**) can be found on the right for each time period (2009/10, 2011 and 2012) . 78
6.6. The different functionality provided by the hardware of the NAO robot (left) and the BIRON platform (right): Speech recognition (1), Image processing (2), Speech synthesis (3), Navigation (4), Object manipulation (5). 79
6.7. **left:** Example for calculating the NAO odometry. **right:** Schema for computing the person angle with the NAO platform. 80
6.8. The programming experience of the subjects of the Bonsai Study in 2012. 83

6.9. The programming environment eclipse as it was prepared for the participants. Image taken from the original screen capture during the study. 84
6.10. The simulated test environment (background right) and the SCXML starter GUI (left front). Image taken from the original screen capture during the study. 86
6.11. The subjects' assessment of the programming functionality of the Bonsai framework. 87
6.12. The subjects' assessment of the applicability of the Bonsai framework. 88

1. Developing Interactive Robots - A First Contact

Never judge a book by its cover.

popular proverb

"Wouldn't it be excellent to have a robot at home that can actually interact with people and help them in their daily lives?" - similar sentences can be found in many theses' introductions over the last years that aimed to introduce the field of *personal robotics*. In fact, robots where envisioned to literally be at the doorstep of every home but the technological gap to enable robots to act in *our* world turned out to be much harder than expected as it has been proclaimed by Gates [37]. The robotics community in return has widely accepted that the application space often is tightly coupled with the design of a robot platform. Examples can be found in many different areas, ranging from aerospace industry to surgery and manufacturing. Personal robotics, which serves as the test environment for the work presented in this thesis, in this regard describes the application space of a robot performing in a domestic environment with human interaction partners.

This research area has gained more attention by different research groups in recent years with an increasing number of interactive robotic systems available for various research scenarios. One example of a robot fitting into this application space was the research platform BIRON [143] at the Bielefeld University in 2005. Another research platform, the Cosero robot of the RoboCup@HOME team NimbRo from the University of Bonn, can be seen on the right in Fig. 1.1.

Despite the look of some of the research platforms, the functionality available to developers of such platforms has increased, which also means that the robots became deployable in more and more scenarios. Given these robot capabilities, the developers started to ask questions like "What else can we do with it?" or "How do we use that?". The answer to these questions are sometimes a little surprising because despite the rather simple appearance of the robots the interplay of software underneath is often more difficult and complex than one would expected from their appearance. A simple process such as enabling the robot to actually see, or better perceive, a human standing in front of the robot for example involves many different sensors and a lot of software.

The research in recent years highlighted an additional challenge when developing interactive robots: Apart from numerous advancements in different research areas that are relevant to the robot platform, such as computer vision, the additional challenge was how to develop methods or frameworks that make all the different pieces of software do something coherent - or even better - make them do something useful for the user.

But making a robot do something useful such as following a person actually comprises two questions. The availability of a software component that is able to detect a person that might be standing in front of the robot does not necessarily mean that a developer can use this component on a robot. And once this software component is running on the system it does not mean that the robot is now "capable" to e.g. follow a person. For that, the "tracking component" of the system needs to be somehow combined with the "moving component".

Sometimes the sheer complexity in terms of number of different software components of a system, which may include a software component that tracks a person in the robots vicinity, hinder or even prohibit the combination of presumably easy actions of the robot because there is no component properly combine them. But to make robots act in more complex environments and make them demonstrate more complex actions, actions that combine many capabilities, it is necessary to find methods for the combination of the platforms capabilities into coherent behavior.

Naturally the early systems that where deployed in domestic environments for human-robot interaction (HRI) had to be equipped with software that allows to e.g. detect persons, identify objects or understand speech commands. Hence, one focus of the robotics community at that time was the integration of such software components into a system which than can perform simple actions.

But with more software and more systems at hand that handle software integration issues (e.g. MARIE [24] or CARMEN [92]), focus shifted towards engineering issues arising after a system has been firstly constructed, which among other things means the combination of the available capabilities of a platform in new or different scenarios.

This shift of focus that was happening inside the robotics community is also reflected by the number of different robot challenges that emerged over the last years and the increasing number of participants. Recent years have seen specialized competitions focusing on certain research areas, e.g. the *Semantic Robot Vision Challenge*[1], or on a particular scenario, e.g. the DARPA Grand Challenge [93] and the Urban Challenge [129] - both focusing on autonomously driving cars. Most recently the DARPA Robotics Challenge is focusing on humanoid robots for disaster response. Apart from that there are broader competitions like the RoboCup that has moved from pure robot soccer and diversified into different

[1] `http://www.semantic-robot-vision-challenge.org/`

leagues covering particular scenarios for robots, such as the Rescue League [61][2] or the @HOME League [130][3].

1.1. From Service to Personal Robots

Given this groundwork for integrating functionality in terms of software into robot platforms and the changing applications from these competitions, the challenge was to take these abilities of a robot and make it operate in a world of humans. To be able to do experiments in this area of application the mobile research robot BIRON (see Sec. 3.1) was used during the research carried out for this thesis.

The BIRON system, along with many other research systems is a result of a collaborative engineering process in an environment of rapidly changing technologies. They consist of a large number of hardware and software components, each of which solve problems from many different research areas (navigation, mapping, perception, planning, speech understanding, dialog, etc.). Personal robots additionally face the challenge to actually act together with humans in a shared environment. Human-robot interaction (HRI) in this regard has taken its steps ahead, moving away from command-oriented interactions to more complex scenarios to further improve the interaction with a human. The role of the human user changes from an operator, as e.g. found in the manufacturing industry, towards an actual interaction partner, e.g. acting in a domestic environment.

As mentioned before the area of application has an impact on how robotic systems are developed and in what way they are optimised. On the left of Fig. 1.1 there are three popular areas of robot applications shown and associated features that are especially crucial for that area. Industrial robots for example (shown in green) are often optimised for one special task that they need to do repeatedly and fast. Hence, stability is important to not interrupt a production line, an example of such a robot is the Titan [4] robot. Following the *International Federation of Robotics (IFR)* [5], a non-profit organisation by robotics organisations to help and promote the robotics industry worldwide, a service robot should provide services useful for humans or equipment that are not manufacturing operations. This can be a transport robot in a hospital (e.g. the TUG[6]) as well as a toy robot for entertainment (e.g. the Pleo [7]). Flexibility in terms of deploying the robot in different situations while being able to adapt the actions is important for service

[2] http://www.robocuprescue.org/
[3] http://www.robocupathome.org/
[4] http://www.kuka-robotics.com/germany/en/products/industrial_robots/heavy/kr1000/
[5] http://www.ifr.org/
[6] http://www.aethon.com/solutions/deliver/
[7] http://www.pleoworld.com/

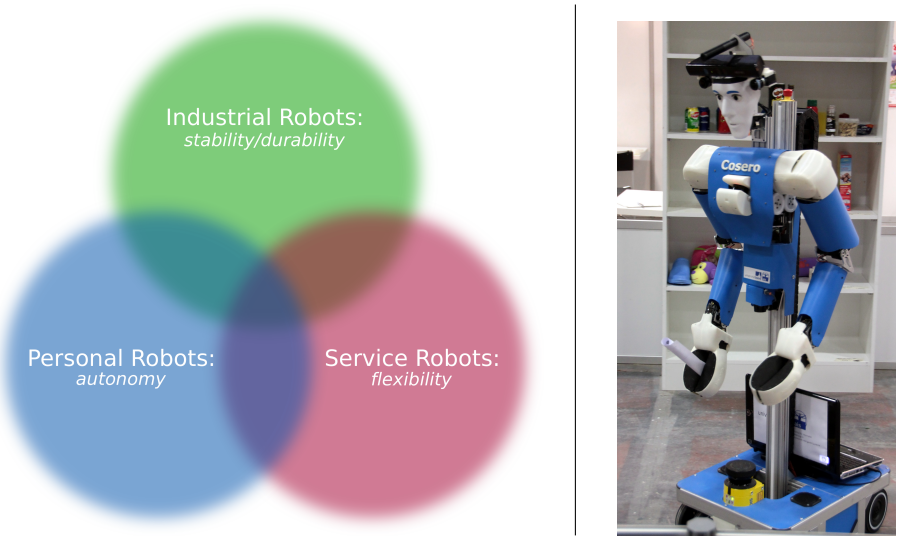


Figure 1.1.: **left:** Different robot applications and resulting requirements for robotic systems. **right:** The robot Cosero of the RoboCup team NimbRo doing the registration task at RoboCup 2011, Istanbul.

robots (see red circle left in Fig. 1.1). A personal robot is to a certain extend a service robot in your home that, apart from fulfilling services for you, is able to naturally interact with people. Because the user does not want to or even is not able to re-program the robot all day, autonomy is a crucial factor for personal robots. They need to be able to take action by themselves based on e.g. their observations of the environment (see blue circle left in Fig. 1.1). Until now such a robot can not be purchased "*of the peg*" but many researchers around the world have platforms to experiment and test such systems (e.g. the PR2 robot[8]). The BIRON platform that serves a similar purpose will be introduced in more detail in Sec. 3.1.

Obviously these features of robotic systems do not only apply in these areas. The optimal robot is stable and flexible and autonomous, which still is a big challenge for developers. But we have seen a growing number of robots entering new areas of application and some are even entering peoples homes (e.g. vacuum cleaning robots) but the lack of robots performing multiple tasks in peoples homes may serve as an indicator that there still needs to be research on how to combine these features and have an optimal robot that could be placed right in the center of Figure 1.1.

[8] `http://www.willowgarage.com/pages/pr2/overview`

1.2. Research Questions

Despite the fact that single capabilities needed to solve various tasks with robotic systems (e.g. person tracking) are well established, robots frequently fail when they need to show these capabilities in a complex real world scenario. There are different reasons for that, one certainly is that it is still difficult to test all necessary software of a robot system under realistic circumstances. Another might be the reliability of results of certain software components, e.g. object recognition, under real world (read: dynamically changing) conditions. One aspect that is typically underestimated is the efficient combination of robot capabilities or coordination (see also Sec. 4).

Given the improvements over the last years in terms of available functionality and middleware (communication between software) for robots, this coordination has become one of the key factors for robots performing in real world scenarios because it allows them to operate more autonomously.

There are various areas of research that are also working on improvements for the performance of robots in real world scenarios ranging from computer vision to psychology. But with the availability of more complex (in terms of number of available functions) and more compatible systems there is also a growing demand of developers that work with these systems in real world scenarios, improving the system on a behavior level rather than improving a single component of the system. This means that there is also a shift of the focus of developers from integration aspects (read: including new software into a system) towards the change and adaptation of system behavior in iterative evaluation cycles.

The availability of systems and the resulting change of focus leads to the following research questions that have been the scaffolding of this work:

Improving Behavior. *How can we improve the development of robot behavior based on experience gained in real world interactions over time? This includes identifying what needs to be changed and how to achieve the change.*

Adaptivity & Reusability. *How can we make it possible to easily re-use robot capabilities in different applications that have been evaluated in other scenarios, on other platforms, or with different communication frameworks (middleware)?*

Enable Coordination. *How can we make behavior coordination, the combination of different robot capabilities, for interactive robots easier and reusable in different scenarios for developers?*

It is important to note that the term *behavior* is ambiguously used throughout the robotics community. A definition of the term for the context of this work can be found in Chapter 4.

All these questions need to be answered when trying to build the optimal robot that combines the features explained in Fig. 1.1. But these questions also indicate that there are two main perspectives on the topic. One is the view of a system evaluator that has to improve the behavior of a robot from interactions observed in the real world. The second perspective is the one of a system developer who may be an expert for one area of the behavior and needs to be able to integrate improvements from the observation into the robots behavior in a reusable manner that it can be further evaluated and improved. These two roles, the one of a system evaluator and the one of a system developer, are a separate problem but they are not necessarily taken by different individuals.

Both of these perspectives also resemble a difficulty encountered in robotic systems: The discrepancy between a desired approach of coordinating the robots behavior and the actual implementation of the system. This means that the implementation of a system can limit the reusability or the ability to combine capabilities of the robot system. This results in more engineering efforts that have to be applied to achieve the desired outcome with a robot.

The guiding principle of this work was to take the two perspectives of the system evaluator and the developer into account and find a consistent solution for the system design and the implementation of robot behavior.

1.3. Contribution & Outline

This work focuses on the engineering aspects resulting from an iterative modeling approach of the behavior of a robotic system and the according challenges for developers of such systems.

The main contribution of this work is a modeling and developing process of robot behavior facilitating the concept of *behavior modules* (see Sec. 4.2.6) that allow developers with little to no experience with the system to adapt and improve these building blocks over time. These concepts where implemented in a modeling framework, named Bonsai, that allowed to evaluate the modeling framework and the according concepts in real world scenarios (see Sec. 6.2) and on different platforms (see Sec. 6.3).

Following up this Introduction, the thesis is structured in five parts: In Chapter 2 I will give an overview of the current state-of-the-art interactive robot systems and explain their approaches to achieve real world interaction, followed by an overview of the used tool set in Chapter 3. After that chapter 4 will give a detailed explanation of the system foundation and the underlying concepts that emerged from the experience gained from real world experiments with a robot system over time. Gray boxes provide additional information on robot scenarios that have been research topics, e.g. the Home Tour Scenario (see Exc. 2.1.4), or that are part of

the RoboCup (see Exc. 6.1.1) as well as additional background information. They do not directly fit into the chapters but do provide interesting information for the reader to get an impression of what actually is going on during RoboCup tests. The implementation details will be exemplified in chapter 5. The work has been intensively tested and evaluated in two main scenarios over the complete timespan of this thesis. The different results from user studies and the RoboCup@HOME competition will be presented in chapter 6. Finally I will conclude and illustrate some future perspectives for interactive robots and their software in the future.

2. Interactive Robots: Software & Systems

All experience is an arch to build upon.

Henry Brooks Adams

In this Chapter I will give an overview of different principles for *system architectures* and introduce some the characteristics implemented in existing robotic systems. In addition to that I will discuss some of the developments in recent years that have led to an increasing number of interactive robotic systems which also fostered the progress in the field of human-robot interaction (HRI) towards more flexible and complex scenarios in open and unstructured environments. Hence, this chapter not meant to give a complete list of interactive robot systems but rather highlight certain developments that are relevant for developing and improving robot behavior. I will give an overview of different robot scenarios that are used to evaluate the performance of a robotic system acting in such a scenario. After that I will introduce principle concepts for control and coordination of such systems, followed by real world examples of systems and tools that are already available. The platform used during this work and the according software will be introduced in the next chapter.

2.1. Scenarios for Interactive Systems

The term *software architecture* is often used and is important for interactive systems since, as also pointed out in [23], it is one corner stone of what a system or robot will be able to do. It focuses on the software components and their interaction. The Institute of Electrical and Electronics Engineers (IEEE) defines a *software architecture* in their recommended practice 1471-2000 [51] [1] as follows:

[1] `http://standards.ieee.org/findstds/standard/1471-2000.html`

Fetch 'n' Carry

A *Fetch 'n' Carry* refers to a typical task of a service robot in which the robot has to fetch a defined object for a user from a known location and deliver it to the user. This kind of setting was subject in various user studies [53, 136] and was also one of the first major tasks in the RoboCup@HOME competition in 2008/2009 [94]. The complexity of the task may vary, e.g. the robot might have to identify the correct object from a set of objects or may get instructions from where to fetch the object ("*Go to the kitchen table!*"), but the common goal is to bring an object from one location to the user. More recently this task also includes autonomous grasping of the object and object recognition in a cluttered scene.

Excerpt 2.1.1: About the *Fetch 'n' Carry* task

> **Software Architecture.** *The fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution.*

For this work the software architecture plays an important role because the modeling and developing of robot behavior is influenced by the interactions of the software components and their organization. An introduction to the topic can be found in [34] However, the principles for software architecture in general are not the subject of this work. More details will be discussed in Chapter 4.

When it comes to comparing of interactive robotic systems, which still is a difficult task, the software architecture and the actual hardware of such systems is often abstracted. The problems are usually solved in simulation or in constricted scenarios as e.g. also pointed out by Baltes [7] and Behnke [9]. This problem of measuring performance of the systems has lead to toy domains (e.g. block stacking or towers of hanoi), which in some parts provided a measure of performance. However, the domain can influence the performance and often covers only few aspects of a system and is an abstraction of the real world.

With these lessons learned, more complex scenarios for interactive robots have emerged. There is a variety of prototypical scenarios for service robots that on the one hand provide the basis to study the systems in real world interactions and on the other hand enable a comparison between different robots or methods in the same scenario. The following scenarios where selected because during the evaluation (see Sec. 6.2 and Sec. 6.1) the robot used during this work had to perform in all of the scenarios or scenarios that where very similar to the ones presented here. Apart from this testbed character the selected scenarios are well established and provide a good basis for comparison since many different robots performed these tasks. The Excerpts give a short overview of what is actually happening in such a task. Additional requirements for the selection of these scenarios were that they

can be conducted in real world environments and that they involved interaction with human users, because this is the target scenario for the work in this thesis.

Each of these scenarios focuses on certain capabilities of the robot to drive development in that area. The *Tour Guide* scenario (see Exc. 2.1.2) for example mainly focuses on the navigation of the robot, popular examples are e.g. [127] and [117]. The *Fetch 'n' Carry* scenario (see Exc. 2.1.1) in contrast focuses on mobile manipulation in complex scenarios, popular examples are e.g. [12] and [126]. A scenario that focuses more on the perception part of the robot, which obviously also plays an important role in all previous scenarios, is the *Search for Objects* scenario (see Exc. 2.1.3), a popular example is e.g. [86]. Generally speaking the development of adequate scenarios (or applications) guides the research in different areas and can help to figure out e.g. where more research effort is needed, making the building of applications an integral part of the research process.

Tour Guide

A *Tour Guide* robot is typically guiding visitors of e.g. museums or labs to predefined positions to give information about the environment to the user. The scenario focuses on simultaneous localization and mapping (SLAM), navigation and obstacle avoidance. Additionally the robot should be able to move and operate robustly in crowded spaces and sense humans. The user interaction is typically kept simple, e.g. via buttons ("Start Tour") or keyboard. The robot should be capable of providing audio-visual feedback to the user to communicate the necessary information and additionally indicate a system status. For that matter some systems also incorporate simulated emotions and/or facial expressions that can be displayed.

Excerpt 2.1.2: About the *Tour Guide* scenario

But this variety of foci also demonstrates a common challenge in robotics research: Before one can investigate an interactive system in a complex scenario, the system itself needs to be capable of navigating, sensing and communicating. This in most cases implies - apart from people doing the software integration from different areas and a suitable hardware platform that can be used - another crucial factor for developing such a system: Time.

Obviously *computing time* on a mobile system with limited resources is always a factor that needs to be considered when developing software. But in this context *time* refers to the effort spent on developing new features or improving existing ones. The time for testing the overall system is also a corresponding factor. Naturally the first implementation of a system will not cover all aspects necessary for the system to perform equally well in different scenarios. The way of investigating such systems will result in a *system architecture* that is at least in parts optimized for a certain scenario, often simply by making assumptions about software, the

available information or the surrounding.

Search for Objects

The *search for objects* scenario tackles the problem of a robot autonomously searching the environment for known objects. This means that the robot may have knowledge of the surrounding, e.g. a prerecorded map, and of the objects to search for. The task is to autonomously search for the objects in the environment and notify in case of a successful match, e.g. by exclaiming the label of the found object. Variations of this scenario where e.g. part of the *Semantic Robot Challenge* as well as of the RoboCup@HOME competition. In the Semantic Robot Challenge the objects have to be learned from the world wide web from a simple text file description of the object additionally to afterwards searching for them in a rather simple but unknown scenario [a]. In the @HOME challenge the object models are trained beforehand but the environment is more complex (domestic home) [95].

[a] `http://www.semantic-robot-vision-challenge.org/rule.html`

Excerpt 2.1.3: About the *Search Environment* task

The engineering issues resulting after a system has been firstly constructed shifts the focus from the integration of components towards the change and adaptation of system behavior in an iterative evaluation cycle. In terms of the *system architecture* the focus is set on the efficient adaptability and the reusability of the behavior of the system.

2.1.1. Coordination: Arbitration and Command Fusion

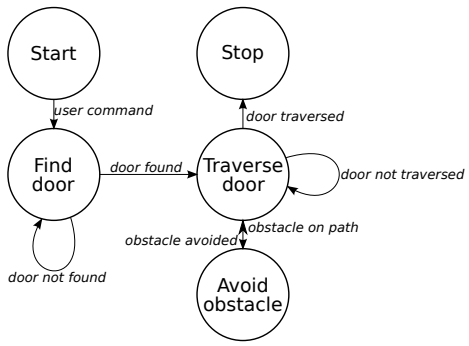


Figure 2.1.: Example from a Discrete Event System (DES), showing a *traverse door* state.

It becomes clear that the aspect of coordination in such systems, as there where discussed in the prior Section, has a tremendous effect on the robot performance. For that reason I will give a short description of what coordination means in the context of interactive mobile robots.

Any system that is confronted with a real world scenario needs to have a mechanism to select and activate an action or sequence of actions to produce a coherent behavior to cope with the current situation. This mechanism, also referred to as *action selection problem (ASP)* as e.g. described by Maes [83], is called behavior coordination.

This problem has been the focus of many researchers from different areas, including ethology, artificial life, virtual reality and others. I will focus on the coordination for mobile robots (physical agents), which is mainly an engineering challenge to utilize robots to perform specific tasks. Following the argumentation of MacKenzie [82] and Safiotti [111], the mechanisms for behavior coordination can be divided into *arbitration*-based and *command fusion*-based approaches. The first class deals with the activation of one appropriate behavior whereas the latter class deals with methods of combining the behaviors that are activated. Popular methods for the arbitration are e.g. *Priority-based* or *State-based* approaches, popular methods for command fusion are e.g. *Fuzzy Logic* or *Voting* approaches. I will focus on *discrete event systems (DES)* because the BIRON platform (see Sec. 3.1) as well as most ROS-based systems (see Sec. 2.2.4) fall under this definition. A detailed overview of coordination mechanisms can be found in [104].

Discrete Event Systems (DES) Discrete Event Systems (DES) is a popular state-based approach for arbitration systems as e.g. described by [65]. In this approach the interaction of the system with its environment is modeled using Finite State Automata (FSA). The selection of the behavior is done via the transitions of the different states. When a certain event is detected, e.g. a person in front of the robot or an open door, the according transition is performed which activates a new state. A state in this case refers to the execution of certain actions of the system to cope with the current situation. Fig. 2.1 shows such a state for traversing a door. This FSA handling the different actions of the system was also called *plant*. Originally there was a second FSA, called the *supervisor*, that can interact and modify the according actions (originally also called behaviors) of the system. However, in practice the modeling of such systems with the supervisor was complex and error-prone since the supervisor had to control all actions and even simple errors within the supervisor can result in restraining or even blocking the execution of any actions.

Before I will give a definition in the next Section under Def. 4 of *robot behavior* in the context of this thesis, we need to take a closer look on the different principles for *Software Architectures* that are in use for Mobile Domestic Service Robots. A general overview can e.g. be found in [85].

2.1.2. Control Principles: Deliberative, Reactive or Hybrid

In this section I will give an overview of the different control principles that can be found in mobile robots. It is important to note that these principles, as well as the coordination mechanisms explained earlier, are independent of the software

architecture of the robot. This means that the same software architecture, which means the same way the software components interact with each other, can realize different control principles and vice versa. It is therefor important to understand what the effects of the different control principles can be and what the possible shortcomings are.

As mentioned previously, there are common principles for structuring the control that are applied on interactive robots that grew out of the necessity to compensate shortcomings of the control architecture in different scenarios. This can be shortcomings of existing structuring principles in terms of e.g. extensibility (or the lack of it) or any other aspect of the architecture that may limit the use of the principle on a robot or in a scenario. This means that the principle suitable for controlling e.g. an autonomous mobile robot monitoring factory buildings is not necessarily suitable for an autonomous service robot interacting with humans. The control architecture from a systems point of view takes care of structuring the control flow, exchange of information in a system to manage the control, and the information flow, exchange of semantic information in a system, accordingly. In general we can distinguish three different principles that have proven to work in different scenarios [98, 3, 22, 32, 40, 72], a good overview can be found in [85]:

Deliberative Systems , sometimes also called *plan-based* systems are based on two main concepts that enable the system to properly act in an environment: Modelling the environment with appropriate sensors and plan the execution of an action according to the information and the world model. In the literature this is referred to as the *Sense-Plan-Act* [16] approach. A system, in a first step, checks the current sensory input (*Sense*) and compare it with the information available from the world model. The knowledge of the world model typically was represented as a set of rules that would apply to a sensory input. After that a plan is computed, based on the available information and the actions from the word model (*Plan*). Finally the actions are executed (*Act*) according to the plan to achieve a certain goal.

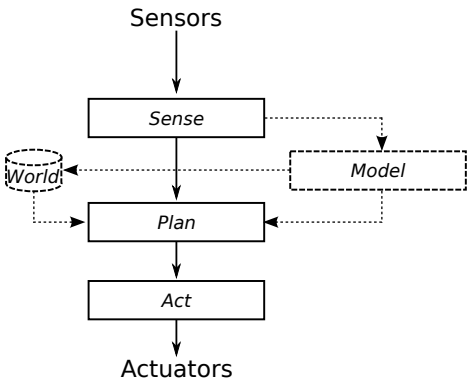


Figure 2.2.: Deliberative System according to *Sense-Plan-Act* and *Sense-Plan-Model-Act* (dashed).

There are a number of pitfalls contained in this way of controlling a robot that make it difficult to facilitate this principle in current real world systems: The set

of rules of the world model are static and typically created by the developer. It is almost by definition inaccurate since it is hardly possible to foresee all situations a robot could encounter. A later introduced variation that tries to dynamically remodel the environment, called *Sense-Model-Plan-Act*, in case of an inconsistency between model and the perceived world, does try to compensate for that. Unfortunately it is still difficult to integrate new knowledge into the world model and it highlights another pitfall, as has been described by Graefe [43]: The computation of a plan takes time. Even with modern computers that are a lot more powerful than 20 years ago the computation takes time, because the complexity of the planned action directly effects the computing time. The implication of that is that the system is unable to react to dynamic changes in the environment. This is due to the breakdown of the different steps that are carried out sequentially and might, in a very dynamic environment, lead to a system that can not act at all because the change of the environment happens faster than the planning and execution of the action. In controlled, non-dynamic environments it is still possible to achieve good results with a control architecture like this. For domestic environments with an autonomous mobile robot for human-robot interaction this model, however, is inadequate.

Reactive Systems in contrast to the deliberative systems do not plan or model the environment. Inspired by biological systems this approach maps sensory input directly to actuators of the system, which mimics the *Stimulus-Response-Model* [64] of biological systems. Because no planning or modelling step is required, the reaction time of such systems is dramatically lower than in deliberative systems. Reactive systems on the other hand loose the ability to plan more complex tasks and can only react to what the system can perceive at a given moment. Such systems serve well in restricted scenarios with a limited set of input stimuli for the robot that can be mapped to the actuators. The mapping happens according to a small set of rules that describe a condition under which a certain action should be executed (*Condition-Action-Pair*).

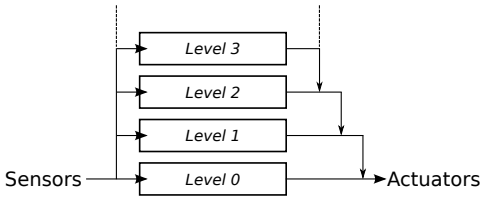


Figure 2.3.: Diagram of a *Subsumption* architecture with 4 layers of competence.

This reactive approach was motivated by the shortcomings of the deliberative control, especially in terms of reacting to dynamic environments. The resulting *subsumption architecture* by Brooks [14, 15] and the later enhancement of the *behavior language* are well known. The additional higher *levels of competence* (see

Level 1-3 in Fig. 2.3) of the architecture allowed to move away from simple, purely reactive systems towards a task-based decomposition of actions that became the beginning of *behavior-based robotics*. This approach does not model the environment but uses the world as much as possible as its own best model via perception. This allows for parallel execution of multiple *behaviors* at once and enables a robot to decide, according to current sensory input, to explore the environment or avoid objects. The pitfalls of this approach lie again in the details, as e.g. discussed by Hartley [49]: As mentioned before, planning ahead is not possible with such systems and the interaction between the different *levels of competence* is static, which means that also for these kind of robots the scenarios in which they can perform is rather limited and changes in the different levels are difficult since it is hard to predict how the changes will influence the actual performance of the system. This also implies that for domestic environments with an autonomous mobile robot this approach is also inadequate.

The Home Tour Scenario

The *Home Tour* scenario started in 2004 as a key experiment of the Cognitive Robot Companion (Cogniron)[a] project with the goal of enabling a robot to learn the topology of a previously unknown apartment and its artifacts, the identity and location of objects and their spatial-temporal relations (see [115]).

The "Home Tour" key experiment was used to demonstrate the dialogue capacities of the robot, and the implementation of human-robot interaction skills as well as the continuous learning of both spaces and objects. To realize this scenario, a robot needs to be mobile, interactive and needs to possess a high standard of perceptual capabilities. The robot must be able to follow a user through the apartment and remember, e.g. via pointing to an object, the robot needs to be able to understand the user's speech, track the gesture and detect the object that the user is pointing to. Additionally, if the user or guide introduces a room to the robot, e.g. by saying "This is the kitchen", the robot should remember the label "kitchen" and mark it in the map. The BIRON platform was used for this experiment and the development paradigm of a close Implementation-Evaluation-Cycle was adopted.

[a] http://www.cogniron.org/final/RA7.php

Excerpt 2.1.4: The *Home Tour* scenario.

Hybrid Systems are a result from the different problems with purely reactive or purely deliberative control. They have lead to different methods, which enable systems to combine reactivity with adaptive execution of plans (see e.g. [27, 35]. Current robotic systems in dynamic environments facilitate a hybrid architecture (see e.g. [73]). The main goal is to enable a system to react to dynamic envi-

ronments whereas at the same time a deliberative component allows to execute more complex tasks [71]. The components realizing this functionality are organized in layers: deliberative, intermediate and reactive layer (see e.g. Gat [36]). The deliberative layer, which typically is on top of the others, generates plans to achieve higher level goals, e.g. navigate to a position on a map (e.g. a room). It is necessary to switch the control between deliberative and reactive components in the according situations, which often is done in the intermediate layer. The reactive layer consists of components that enable the robot to compensate for dynamic changes in the environment, e.g. obstacle avoidance, and can execute simple actions that need no planning, e.g. following along a wall.

Even though it is possible to perform many different kinds of tasks with a system like this, e.g. driving to a location on a map (deliberative) while avoiding dynamic obstacles (reactive) on the way, the major strength is also a problem with these kinds of systems: Switching the control. This means that *hybrid architectures* need to switch the control of either the whole system or parts of the systems between reactive and deliberative components. Typically a centralized component (the *sequencer*), as also described in [22, 36, 99], handles the switch. In Sec. 3.1 two different iterations of such a sequencer for the research platform used in this work are illustrated. This component is crucial for the coordination of any *hybrid system* as it influences all tasks executed on the platform and with extending or changing the behavior of the robot, this component as well needs to be adapted. Even though *hybrid systems* are adequate for the work presented here, the *sequencer* problem needs to be tackled. The steps taken to handle this problem in this work are described in Sec. 5.2.

2.2. Robot Architectures in the wild

In this section I will introduce tools or libraries that relate to my work due to their application, e.g. service robotics, or due to their methods. There are many robots out there but many of them share the same principles or even the same framework underneath which would make it unfeasible to list them all. For that matter I have decided to only address work that has a similar approach or contributes to a similar research topic as the Bonsai framework developed during this thesis. For the relevant aspects I will also explain why certain frameworks where not suitable for this work. The main reason for this is to provide an overview of tools and methods that try to tackle the problem of modeling and/or implementing robot behavior, thus, providing a context for comparability of the work presented in Sec. 4 and Sec. 5.

As I said earlier many of todays research platforms can be classified as *hybrid systems* that manly differ in three ways: How is the control of the system organized

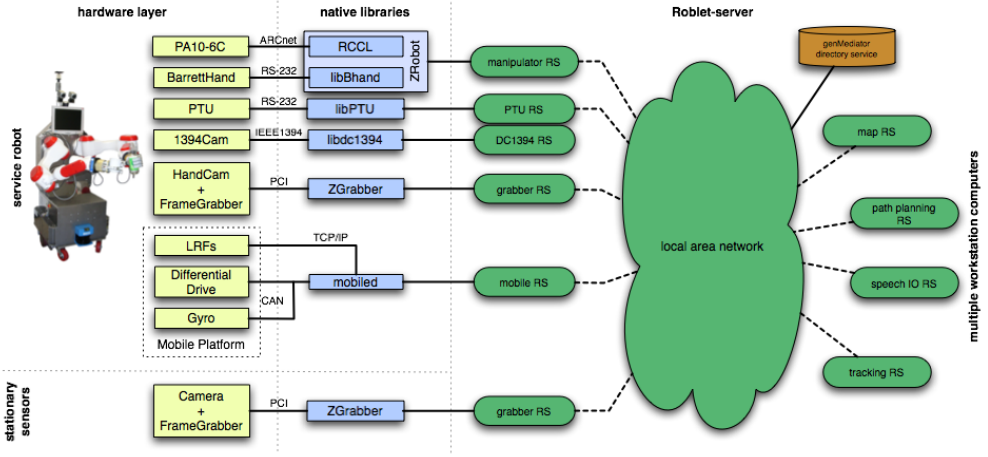


Figure 2.4.: The software architecture of the robot TASER where Roblet-servers (RS) are used to provide a hardware abstraction layer, taken from [6].

(e.g. reactive, see Sec. 2.1.2), how do components share their information (software architecture) and what kind of abstraction levels exist for higher level programming (see coordination Sec. 2.1.1), which are typically the main feature to distinct robot architectures in the wild. I will begin with a framework for higher level programming of robots.

2.2.1. Roblet®Technology

The Technical Aspects of Multimodal Systems (TAMS) [2] group at the University of Hamburg has published a software framework to ease the development of *high level* applications for mobile robots. The so called Roblet®technology by Baier [6] is a *client-server* based middleware in Java that mainly was developed for the robot TASER (left on Fig. 2.4) that also is developed at the TAMS group. The main goal of these Roblets®is to provide higher level functionality of a certain hardware via sending a request, containing an executable Roblet®, to a Roblet®-server. Because a Roblet is well defined in this context, similar to e.g. a Java applet that is run inside a browser, the server can directly execute the request and thereby reply to the request sent by the client. Because of the *client-server* infrastructure Roblets can work in distributed systems. Roblets are subdivided into *Modules* and *Units*. The Roblet®mainly serves as container that encapsulates the network communication and execution on a server. The so called *Modules* extend a Roblet®server to encapsulate a class of similar functionality for a specific

[2] `http://tams-www.informatik.uni-hamburg.de/`

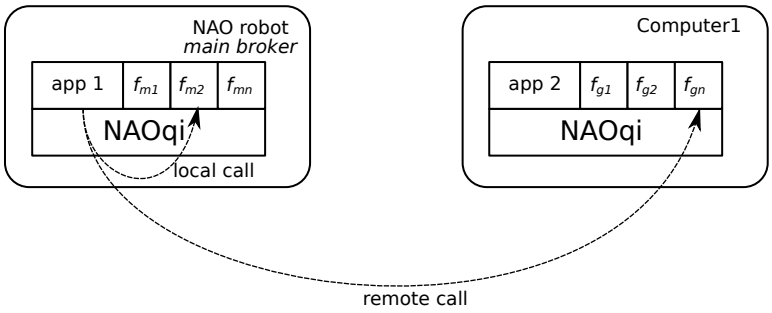


Figure 2.5.: Example of two applications (app1/app2) running distributed on the NAO robot and on computer1. Applications can use all functions registered to the main broker (local/remote call).

hardware. In case of the TASER robot this was e.g. a *Speech Module* or a *Pan-Tilt Module* to control the according hardware and functionality. *Units* are Java interfaces inside the *Modules* that allow for an abstraction of the native hardware interfaces. To request the current pose of an arm of the TASER robot (see Fig. 2.4 top left), a *Module* must implement the according *Unit* that will provide a function to get the current pose of the arm. As depicted on the right side of Fig. 2.4, the Roblet®servers (RS) on the left (e.g. *grabber RS*) provide the hardware abstraction of the robot whereas the servers on the right (e.g. *tracking RS*) provide higher level functionality on this hardware. It is important to mention that only some of the servers are running on the actual robot, e.g. the *path planning RS* runs on an external computer connected to the same network.

This Roblet®technology is relevant to the work presented in this thesis because it aims to provide a task-based abstraction for developers that provides higher-level functionality of a system. Even though the level of abstraction is based on the robot hardware, the execution environment resembles a Hardware Abstraction Layer (HAL) of the TASER platform (see also [137]), Roblets®provide a directly executable entity (Java classes). This is similar to the approach of the Bonsai framework (see Sec. 5.2). However, the focus on hardware abstraction in combination with the restricted execution environment and a missing higher-level control abstraction layer made the Roblet®technology unsuitable for the work presented here. Additionally, the *client-server* based approach induces a high coupling between the Roblets but does not provide a solution for the *sequencer* problem.

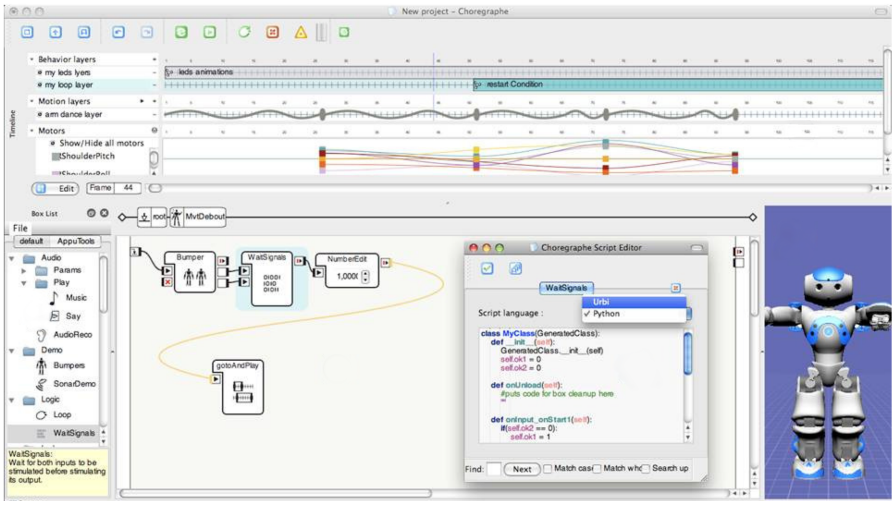


Figure 2.6.: A screenshot of the Choreographe User Interface.

2.2.2. Graphical Tools: Choreographe and NAOqi

One of the robots that has had a big impact to the community in recent years is the NAO robot (see Fig. 6.6 left) developed by Aldebaran Robotics [3]. The NAO is a small and relatively inexpensive humanoid platform (see [42]) that was chosen as platform for the RoboCup Standard Platform League [4] in 2008 and is present in different research labs around the world and features in various research projects (e.g. ALIZ-E [67][5]). Apart from the appealing appearance of the robot the availability of a standardized middleware and programming environment have lead to the wide spread adoption of the platform.

This wide adoption of the platform has put the manufacturer into a difficult position in terms of programming the robot. On the one hand the manufacturer wants to avoid to give a detailed insight into the platform and the hardware but on the other hand wants users working with the platform to be able to quickly put together movements or behavior of the robot. This means that users with possibly no programming experience need to be able to tell the robot what it should do. At a first glance this challenge sounds similar to some of the research questions presented in Sec. 1.2, which is why I will introduce the NAO software in more detail here.

There are two main tools for the robot that I will introduce: The graphical programming tool *Choreographe* (see [106]) and the robot framework NAOqi. The NAOqi framework mainly consists of two parts, namely the middleware core and

[3] http://www.aldebaran-robotics.com/
[4] http://www.tzi.de/spl/bin/view/Website/WebHome
[5] http://www.aliz-e.org/

different functional components (modules) that were specifically developed for the NAO platform (e.g. a color tracker or text-to-speech).

The middleware allows to have distributed binaries, providing functionality for the robot, which are called *modules* that are registered to a main broker instance which manages the different *functions* of the modules. This means that an application running on the robot can use the *walk()* function provided by the *Motion* module as well as a custom function provided by an application running on a computer (see Fig. 2.5). Some essential modules are run directly on the robot, e.g. the *Motion* module, whereas functions that need more computing power than available on the NAO platform can be run on external computers. It is also possible to have multiple robots controlled from one external computer, as it is e.g. done for the RoboCup standard platform league.

To make the rapid prototyping of e.g. robot movements easier the Choreograph tool is also provided by Aldebaran. As a matter of fact, Choreograph itself is a special NAOqi instance that can be run on a computer that provides a graphical user interface to compose sequences of movements and speech as well as a small preview of the composed actions in a small simulation window (see Fig. 2.6 top: time line, left: available NAOqi functions, middle: graphical composer, python script editor, right: simulation preview). A behavior in NAOqi is a piece of software that controls the robot using the NAOqi module functions. Choreographe provides these functions in a graphical manner, e.g. detect a bumper pressed or produce a speech output. The combination of these actions is enabled via the component model of the modules which have to implement the NAOqi module interface and use the request-reply-based communication provided.

We have seen that the graphical interface helps when it comes to programming the robot by unexperienced users. The NAOqi, however, is first and foremost a platform specific middleware to integrate software components into the NAO robot. The graphical interface alone hardly serves as a behavior modeling approach and makes clear that a distinguishing between programming functional software components and the behavioral layer (see e.g. Fig. 4.1) is needed. Lastly, NAOqi only suports the NAO robot and its hardware which makes it impossible to use for any other robot platform.

2.2.3. The Behavior Markup Language (BML)

An approach that, in contrast to the previous tool, explicitly tackles the modeling of behavior is the Behavior Markup Language (BML) [63].

It is an XML-based description for modeling the verbal and nonverbal behavior of humanoid agents, more specifically so called *Embodied Conversational Agents (ECA)*. It is one result of an effort to standardize behavior and functional languages and focus on the similarities that existed among earlier approaches (e.g. [107]).

```
1 <bml>
2   <gesture id="g1" type="point" target="object1"/>
3   <body id="b1" posture="sit"/>
4   <gaze target="object2"/>
5 </bml>
```

Listing 2.1: BML example for a behavior including pointing gesture (line 2), a sitting body (line 3) and a gazing towards an object (line 4).

The main function of the BML is to describe the behavior of a humanoid character and to allow the synchronization of those behaviors. This focus allows to specify *elements* to describe what the agent should do. These *elements* are e.g. parts of the body (`head`, `torso`, `legs`, `lips`...) as well as actions the agents can take (e.g. `speech`, `gesture` or `gaze`). In List. 2.1 a simple behavior is modeled where the humanoid agent points at a target (`object1` in line 2) while the agent is sitting (line 3) and gazing at another object (`object2` in line 4).

In the second example in List. 2.2 the behavior is extended with synchronization data (e.g. `wait`) to structure the actions of the behavior. These extensions, that can be referenced among each other in a BML document, are called *synchronization points*.

```
1 <bml>
2   <gesture id="g1" type="point" target="object1"/>
3   <body id="b1" posture="sit"/>
4   <wait id="w1" condition="g1:end AND b1:end"/>
5   <gaze target="object2" start="w1:end"/>
6 </bml>
```

Listing 2.2: Extended BML example with synchronisation (`wait`) of the gazing (line 5).

BML does provide a basis to describe behavior for virtual humanoid agents. A BML parser is available in Java [6] and projects that e.g. deal with the modeling of multi-modal interaction with virtual human agents. However, for real world systems the implementation of both the behavior and the synchronization has to be done individually for each system. In contrast to e.g. the previously described Roblets, BML does not provide executable software entities. Until now the available implementations neither provide a control abstraction (e.g *state machines*) nor event processing, which makes it difficult to use BML on a real world system. Thus far, the BML language has missed the important factor of matching the described behavior (XML) onto a real platform. An important step that holds additional engineering challenges that are often underestimated and can even break the described model of a behavior. However, a standardized and human readable format to describe the robot actions is desirable.

[6] `http://sourceforge.net/p/saibabml/wiki/Home/`

2.2.4. The Robot Operating System (ROS)

Unquestionably on of the biggest impacts to the robotics community in recent years had the Robot Operating System (ROS) [109], provided by *Willow Garage* [7]. It was started as an effort to bring together existing tools and libraries from different research areas that are relevant for robotic platforms into one framework that could foster collaboration and standardization.

ROS [8] is an open-source collection of libraries and tools that aim to support developers of robotic systems in various areas of expertise. Amongst them ROS provides a hardware abstraction, device drivers for sensors, manipulators and platforms, libraries such as the computer vision library *openCV* [9] and the 3D image processing Point Cloud Library (PCL) [10], visualizers, standardized message-passing and package management. ROS inherited a lot of drivers and tools for navigation from the predecessor Player/Stage project [11] and added apart from software components from all over the community, a consistent communication (*ros_comm*) and packaging framework.

Software within ROS is organized in three main units: *nodes*, *packages* and *stacks*. Any software that wishes to provide functionality to ROS needs to implement a rosnode at the lowest level, which will basically provide communication with the ROS system. A node therefore is a process within the system that can communicate with other nodes and can be combined to fulfill a certain functionality. A collection of such nodes and possibly other software, configurations or datasets that combined provide a certain functionality is organized as a package in ROS. This can be any functionality needed for a robotic system, e.g. device drivers for a camera (*camera_drivers*). Packages are designed to provide easy reusable code. For higher level functionality these packages can be combined to a *stack*. A stack collects packages to provide a combined functionality. Popular stacks are the ros_comm stack, the navigation stack or the manipulation stack. In contrast to e.g. libraries a stack can be run and provide necessary information without linking against the stack. A detailed comparison of the communication features of ROS vs. other middleware can be found in [138].

ROS has become a de-facto standard for integrating software components into a robotic system and also provides tools for developing robot behavior. Since the focus of this work is the development of robot behavior, it seems like ROS would be an obvious choice to use and extend. Hence, I will give a more detailed introduction

[7] `http://www.willowgarage.com/`
[8] `http://www.ros.org/`
[9] `http://opencv.org/`
[10] `http://pointclouds.org/`
[11] `http://playerstage.sourceforge.net/`

into the ROS SMACH [12] library which was developed in 2010 and why I did not use it. SMACH is used to create robot behavior in a rapid prototyping way and is based on the concepts of *hierarchical state machines, (HSM)* [147], which describe the capability of a state machine to allow the nesting of superstates which are complete state machines for themselves.

SMACH

The SMACH library, as e.g. described by Bohren [12], is a task-level library written in python for rapid prototyping of different robot scenarios. The evaluation system [13] was a small robot platform that was able to deliver drinks and take orders without speech interaction. The system was completely done with ROS and SMACH.
Generally SMACH distinguishes two main concepts: States and containers. States, which are derived from the HSM states, is a local execution of a state and corresponds to the according system performing a certain task. A state always provides a certain *outcome*, which normally represents the result of the processing inside a state. This generally allows developers to model the robot actions with states. SMACH provides different state classes to model certain functionality: The base interface is called `State`, `SPAState` defines a normal state with a predefined set of *Outcomes* (succeeded, preempted, aborted), `MonitorState` blocks the execution as long as a predefined condition holds, `ConditionState` is executed when a predefined condition is true (*Callback*) and `SimpleActionState` acts as a wrapper for the ROS actionlib [14] by simply calling a certain action available from the ROS actionlib.

Figure 2.7.: ROS smach

Containers in contrast provide an execution semantics, which means a container defines how (multiple) states are executed. Available containers in SMACH are *StateMachine* to directly execute a certain state machine, *Concurrence* allows to simultaneously execute states, *Sequence* always executes the states in a predefined sequence (less flexible than *StateMachine*) and *Iterator*, which is very similar to *Sequence* but can auto-generate transitions of a set of states. Fig. 2.8 shows a simple state machine with the states *FOO* and *BAR*, which can be chosen

[12] `http://www.ros.org/wiki/smach`

[13] SMACH has been evaluated during the students project "*Silent Butler*" was part of the ISY practical in 2011 at Bielefeld University. I would like to thank all participants of the project, especially Patrick Renner, Lukas Kettenbach, Phillip Dresselhaus and Manuel Baum for their hard work.

[14] `http://www.ros.org/wiki/actionlib`

from the available SMACH states as explained above. Both states are added to a *StateMachine* container which then is to be executed. In this example *BAR* will be repeated until a condition defined and checked in *FOO* is fulfilled (e.g. execute *BAR* 3 times).

With these building blocks at hand it is possible to construct a robot scenario while taking advantage of the ROS features and libraries. However, I would like to point out some of the aspects that emerged during the evaluation project. SMACH does a great job in combination with the ROS tools to setup a small system and get it running. The SMACH viewer [15] supports the developer with visualizing the currently running behavior and highlighting active parts at runtime. Even though the HSM modeling part of SMACH is a standalone library that is decoupled from ROS, for actual prototyping of a system SMACH introduces a high dependency with ROS on a rather low level. The usage of ROS messages inside the modeled states allow for type and configuration checking (via the ROS tools) at startup, but makes it difficult to reuse such behaviors outside a ROS context. It is possible to send untyped *user data* between states but information produced here is difficult to propagate into the rest of the system. System components are by default locked out of control flow information between different states. SMACH supports exceptions of different states but if not properly handled the complete state machine freezes which in the end means that the robot freezes. The states processing is done in *while()* loops, but it is difficult to interrupt these from external input which can make implementation of reactive behavior a quite difficult.

The available Containers in SMACH already indicate that the tasks that a robot should fulfill are modeled in a rather controlled manner, which means what the robot does (behavior) and how it should do it (*control flow*) are modeled together in the same construct. This is helpful for rapid prototyping scenarios since there is only one place where to code can be changed or added in a short testing cycle. For reusability however this is a major drawback. These aspects will be further discussed in the next section 4. A detailed comparison of the framework developed during this thesis and SMACH can be found in Sec. 7.1. The mentioned drawbacks here and the rather late availability of SMACH (2011) have lead to the

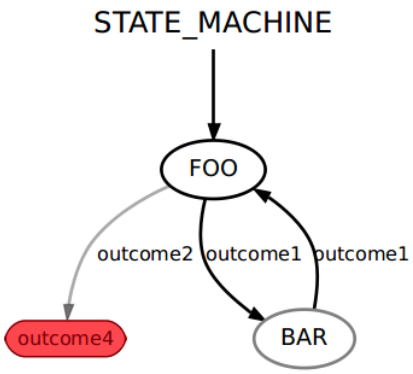


Figure 2.8.: Example state machine with smach *outcomes*.

[15] http://www.ros.org/wiki/smach_viewer

decision to not use SMACH.

2.3. Summary

To summarize this chapter I will shortly discuss some of the issues and challenges that emerged from the related work presented here. We have seen that there are a number of systems and according architectures out there. For a mobile interactive robot that comprise an increasing number of capabilities a *hybrid architecture* was utilized because it can cope best with the different scenarios, as e.g. introduced in Exc. 2.1.4 and 2.1.3, that require reactive and deliberative components.

For the *behavior coordination* (see Sec. 4) of such an architecture, as explained with the *sequencer* for *hybrid architectures*, a solution that avoids such a single sequencer component needs to be considered. Additional support for different mechanisms for coordination (see Sec. 2.1.1) is desirable.

We have also seen that there are a number of different tools and libraries available, but adequate behavior modelling combined with the matching between the descriptive model and executable entities seems to be still difficult. A solution that enables this combination and at the same time supports the usage of existing frameworks for middleware (e.g. ROS) would also be desirable. Not only because this would reduce the development time for a system, but because existing components could be used, and it would also allow a better comparison between frameworks or platforms. In the next chapter I will give more details on the principles that helped to create a solution for these issues.

3. The Tool Set: Platform and Communication Framework

In this chapter I will introduce the robotic system BIRON which has been continuously developed before and during this thesis, making it an interactive robotic systems that has been continuously improved over time span of over a decade. It serves as a experimentation tool in various scenarios for the concepts that will be presented in this work. During this thesis there have been many occasions where the robot was evaluated in real world scenarios. Apart from user studies in a real world apartment (see [76]) the system has been part of the RoboCup Team of Bielefeld (ToBI) [131][1]. After that I will give a short overview of the middleware framework that was used on the platform and the software architecture the solution developed in this work is based upon.

3.1. Bielefeld Robot Companion (BIRON)

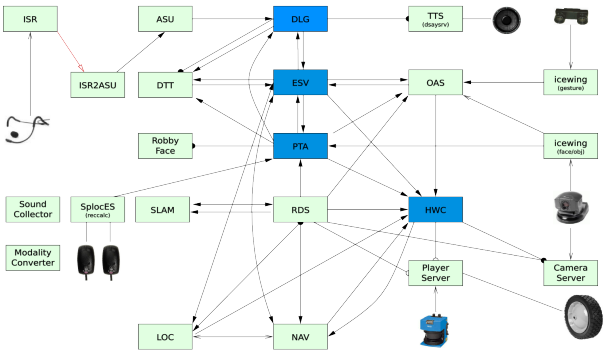


Figure 3.2.: BIRON software architecture with central *Execution Supervison (ESV)*

Certainly there are a number of difficulties for service robots in real world environments that need to be overcome. But to develop robots that can be accepted as *personal robots* in our homes two additional factors play an important role: The robot must be accepted as a *communication partner* by the human user and it must be able to sense and act in an environment that was adapted to humans. To afford research on these topics and the resulting requirements, the Bielefeld Robot Companion (BIRON) was developed at the Applied Informatics Group of the Bielefeld University in 2004.

[1] ToBI website: `http://www.cit-ec.de/ToBI`

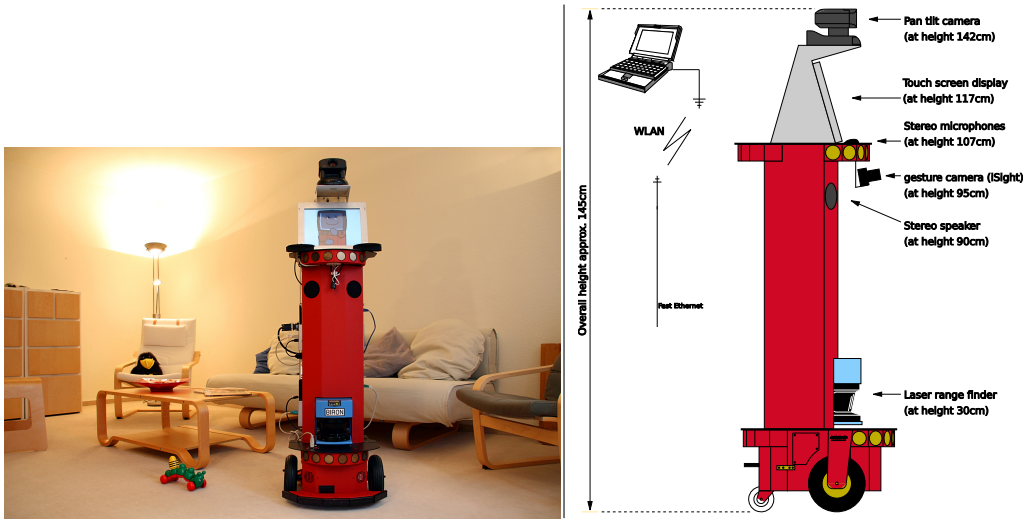


Figure 3.1.: **left:** The 2007 BIRON system waiting in the living room of the real world apartment of the robot. **right:** Schematic view of the BIRON system with its components.

The first version of the BIRON platform was designed for Human-Robot Interaction (HRI) in a real world scenario (see Fig. 3.1). For that purpose, an apartment in Bielefeld was permanently rented for carrying out experiments with the robot. A ground view of the apartment can be seen in Fig. 6.1.

As one robot platforms that was continuously developed over a decade at the time when this thesis was finished, it is evident that there have been changes in the architecture and the hardware over time. I will give an overview over the relevant parts and describe what can be called the evolution of the BIRON platform.

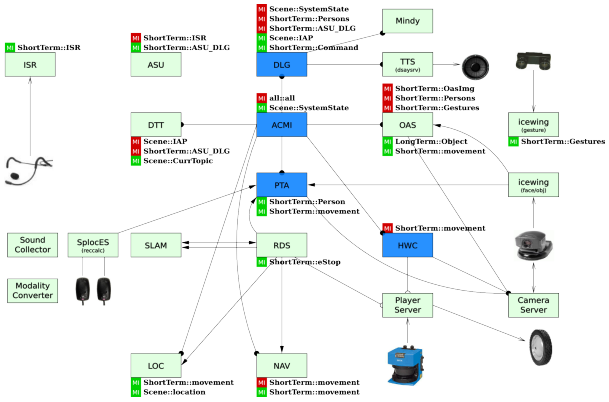


Figure 3.3.: Updated BIRON software architecture with *active memory* enabled.

Influenced by the work of Gat [36] on three-layered architectures and Rosenblatts [110] work on a fine-grained layered architecture for controlling a mobile robot, the architecture of the first BIRON platform was described as a hybrid architecture with a central *Execution Supervisor* acting as a controller and sequencer

Figure 3.4.: **left:** The 2010 BIRON system waiting in the living room of the real world apartment. **right:** The 2011 BIRON system and its components on the right.

for the different components of the system (see [143, 44] for more details). While the early Software Architecture was partly based on an *event-based* data flow with XML messages, the architecture itself was designed as a *three-layered architecture* as described by Gat. Thus, it divides the system into deliberative, intermediate and reactive parts.

The main focus, that was achieved through the *execution supervisor (ESV)* component, was to enable a coherent execution of different tasks and control the different system components from the different architectural layers. This allowed to decouple the execution from a planner component in contrast to other systems at the time, enabling the system to perform tasks or combine reactive and deliberative tasks faster. The resulting highly coupled system as it can be seen in Fig. 3.2, indicated by the direct connections between system components, shows the complexity of the ESV-system. Almost all information from the different system components had to be send directly to the ESV to enable to switch the control between components. Additional connections from components to the ESV where necessary to actually switch the control. This obviously introduces a high complexity to the system, makes changes in the robot behavior difficult since they must be performed in many different places of the system and impose system components with the necessity to store information about the current overall system state.

To overcome these aspects and particularly reduce the complexity of the system,

Figure 3.5.: The evolution of the BIRON system from approx. 2002 (left) until 2012 (right).

more precisely to reduce the necessary connections between components, an *active memory* [135] based approach was introduced that basically switched the control (sequencing) between the different components in place where the necessary information was stored: In the memory (*active memory*). The resulting reduction of connections can be observed in Fig. 3.3, more details can be found in [122]. A more detailed discussion can be found in Sec. 7.1

3.2. BIRON Hardware

A robotic system that is developed over such a long period of time and, more importantly, is deployed in changing scenarios like the BIRON system involves a number of hardware modifications over time. I will introduce the platform and focus on changes/extensions that had an impact to the behavioral spectrum of the robot, which excludes repairs or upgrading the on board laptops. In general it is save to say that the availability of more computing power on board will increase the components that can be run in parallel, which results in more functionality available at the same time on the robot. It does not automatically improve the performance of the robot itself.

The early versions of the BIRON system, which was before the year 2005, are not directly related to the work in this thesis. Hence, the hardware description comprises only the advancements of the hardware from 2007 to 2012. The BIRON system with its components in 2007 is depicted on the right in Fig. 3.1, the left image shows the robot in the living room of the real world apartment that was also used for the user studies described in Sec. 6.1. A detailed descriptions of the hardware and the study in 2007 can be found in [76]. For the following years,

2009-2012, the team description papers of RoboCup team ToBI provide a good overview of the changes of the hardware (see [131, 132, 133, 134]).

The actual robot platform is based on the research platform *GuiaBot*® by adapt mobilerobots[2]. It was customized and equipped with sensors that allow analysis of the current situation in a human-robot interaction and with two piggyback laptops running Linux to provide the necessary computing power. The main sensors of the platform can be seen on the right of Fig. 3.4: Cameras for face/object detection and recognition, a microphone for speech recognition, a 3D sensor for grasping/-navigation and gesture detection, a robot arm for manipulation and a laser scanner for mapping of the environment. An overview of the different changes of the hardware of the robot BIRON from 2002 until 2012 can be seen in Fig. 3.5.

Imaging system: The imaging system plays an important role for the behavior of an interactive robot. Most noticeable in the appearance of the robot BIRON is the change from a pan/tilt camera towards multiple high resolution cameras on top of the robot. This is mainly due to the fact that the higher resolution allows for better feature computation on cropped images for object recognition. Multiple cameras additionally allow to have one camera permanently looking for objects and another one permanently looking for faces. Another extension of the robot BIRON that had a big impact to the behavior, and probably to robotics community as such, was the development of the kinect® sensor (e.g. described by Shotton [116]). With a huge community in the background, software for 3D sensing that uses this sensor was available to the community and was, due to the mass production, very affordable compared to other sensors on the market. In 2011/12 almost all robots in the RoboCup@HOME league where already equipped with a kinect sensor.

Manipulation: Manipulation is essential for a domestic service robot and despite the increased difficulty of manipulation on a mobile platform, it has become a mandatory aspect not only in the RoboCup@HOME challenges. The BIRON platform therefore was extended with a manipulator, a Katana IPR 5 degrees-of-freedom (DoF) (see right in Fig. 3.4, 2nd from bottom). This is a small and lightweight manipulator driven by 6 DC-Motors with integrated digital position encoders. The end-effector is a sensor-gripper with distance and touch sensors (6 inside, 4 outside) allowing to grasp and manipulate objects up to 400 grams throughout the arm's envelope of operation.

Both, the improved imaging system and the manipulator greatly extended the behavioural repertoire of the robot platform but also introduced additional components and complexity into the overall system.

[2] `http://mobilerobots.com/`

3.3. An Active Memory for Information-driven Integration

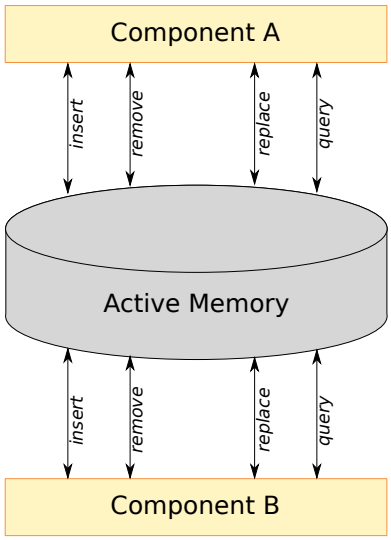


Figure 3.6.: Conceptual drawing of **component A** and **component B** communicating via an *active memory*.

The BIRON platform uses an integration framework that basically puts foreword two main elements that enable an *information-driven integration* approach, which means the integration of system components is data-driven and not bound to explicit links between components of a system. The framework assumes a *component-based system*, as e.g. described by Brugali [17, 18], where every piece of software that implements a functionality of the robot is considered as a *component*. These components share a common interface for execution and information exchange. Typically all components of a system use the same framework, in case of the BIRON platform this was the XML-enabled Communication Framework (XCF) [30]. This framework is based on XML [3] representations of the data exchanged between components as a document-oriented data model. In particular it comprises an *active memory (AM)* as a central integration broker for managing shared data of a system. Conceptually, all the information that are generated by system components is shared through an *active memory*, where it can persistently be stored and retrieved from. Because the data is encoded using XML, *components* can subscribe for particular information via the XML Path Language (XPath) [4], a query language for XML documents, and get notified whenever this subscribed memory content changes or is inserted into an *active memory*. As it is shown in Fig. 3.6 the according operations on the memory are *insert*, *remove*, *replace* and *query*. The *active memory* itself can be seen as a system-wide tupel-space, providing managed read-write access to all system components using the above operations.

[3] Extensible Markup Language (XML) is a markup language for encoding documents in a format that is both human- and machine-readable. `http://www.w3.org/XML/`

[4] `http://www.w3.org/TR/xpath20/`

4. The System Foundation & Concepts

> The belief that complex systems require armies of designers and programmers is wrong. A system that is not understood in its entirety, or at least to a significant degree of detail by a single individual, should probably not be built.
>
> Niklaus Wirth

In this chapter, I will give an overview of the development process when facilitating *software intensive systems*, such as these described by Wirsing [139], after such a system has initially been assembled. I will exemplify the resulting challenges and also give a short introduction into architecture principles and requirements for mobile interactive robots. After that I will illustrate the concepts of the Bonsai framework that are related to these aspects and that can help developers to better fulfill specifications of an architecture. I will start with defining the term *robot behavior* in the context of this thesis to better understand the general idea behind the developed framework.
Like many other research areas, the field of robotics comprises many disciplines and areas that often use a similar vocabulary, even though they are not necessarily talking about the same thing. The context for the systems that are discussed as part of this thesis (see Sec. 2) are *domestic service robotics* in complex environments. This means that challenges a robot faces in this area range from having to follow a person, introduce a robot to a new home environment (see Exc. 2.1.4), delivering tasks (see Exc. 2.1.1) to autonomously searching for objects in an environment (see Exc. 2.1.3). But this also implies, since we speak of domestic environments, that the robot needs to be interactive in terms of being able to perform in a human environment and interact with humans at any time.

Nevertheless the focus of this work is to help developers to create reusable robot behaviors and improve a system over time via real world experiments. Therefore I am introducing the term *robot behavior* in more detail, due to its ambiguous usage in different fields of research. Generally speaking anything that describes

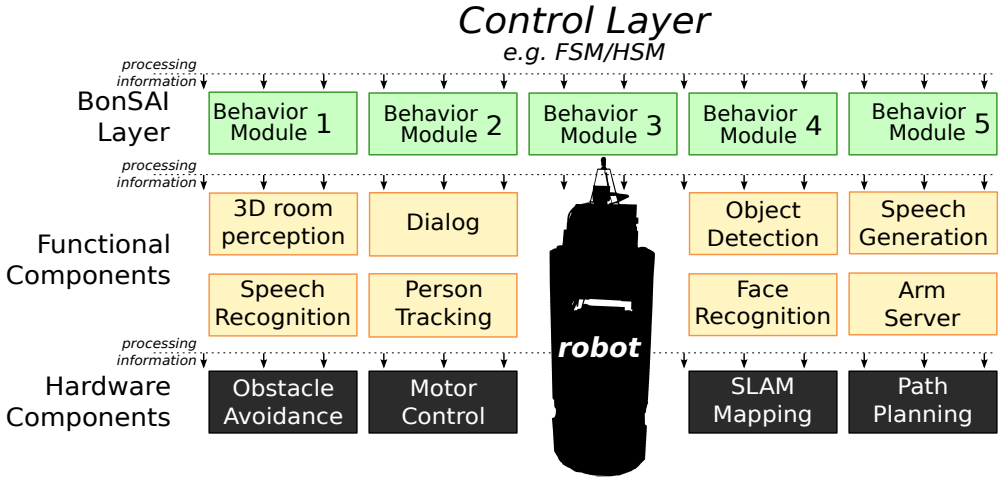


Figure 4.1.: Schema of the *Bonsai Architecture* style in more detail: Each layer processes information generated in layers underneath or in the same layer, enabling a semantic decomposition of the components. Only selected components of the *functional components* layer of the BIRON system (see Sec. 3.1) are shown.

the performance of a robotic system can be considered as behavior. In contrast to the classical *behavior-based robotics* as e.g. described by Arkin [4] I will take a slightly different perspective in this thesis. Arkin describes behavior as a formula that matches a certain input stimulus to a system reaction (see *motor schemas* in [2]). This interpretation works well for various scenarios where the input stimulus can be described beforehand, for mobile robots in a domestic scenario this may not always be attainable. Most importantly this interpretation is only fixed on the robot and excludes the user. For dynamic and interactive scenarios a definition should consider the effect of the robot action onto a user that will interpret the behavior in its own context.

Hybrid research systems, as explained in 2.1.2, are typically implemented via *software intense systems*, which means that a larger number of components is running distributed on multiple machines. In contrast to behavior-based systems, this approach makes use of many pieces of software which are providing a certain set of functionalities to the system, as depicted in Fig. 4.1 (*functional components*). The layering approach, which mainly aims at information hiding between the different layers, is a well accepted approach (see Sec. 2.1.2) and therefore will be supported. An overview of reasonably recent frameworks can be found in [11, 90].

This implies that many components contribute to the overall system behavior in a different way, depending on the current situation. Known behavior-based systems, a good overview can be found in chapter 4 of [4] (pages 140/164-165) and

more recent examples are [31, 73], have covered a smaller subset of these functions, such as navigation and avoiding obstacles, and therefor exclusively influence the execution on such systems. For modern robotic systems this constricted interpretation of *behavior* is inappropriate, because systems incorporate more functionality and there are many components that need to influence the execution of behavior. For example, a navigation subsystem [1] as e.g. provided via the Robot Operating System (ROS) (see Sec. 2.2.4), that is part of many modern service robots [2] already covers the complexity of some former complete systems (e.g. [40, 112]). The interplay between task-level and lower level actions of a robot, comparable to the *subsumption architecture* described in 2.1.2, has become more complex and influential on the overall system performance.

Hence, it is necessary to distinguish between a hardware- and a functional layer where all components are individual software entities running distributed on the robot. The *hardware components* layer (see Fig. 4.1 bottom) contains software components that are dependent on a direct connection to the hardware because they need to run in real time and/or require high data rates, e.g. the Obstacle Avoidance or the Simultaneous Localisation and Mapping (SLAM). The *functional components* layer (see Fig. 4.1 middle) includes all software that needs to incorporate data from either components of the layer underneath or the same layer or does not rely on a direct hardware connection, e.g. the person tracking. Please note that only some selected components of the *functional components* layer of the BIRON system (see Sec. 3.1) are shown. The *Bonsai* Layer includes all the *behavior modules* (see Sec. 4.2.6) of the robot that collectively describe the behavioral spectrum of the robot.

Accordingly the term *robot behavior* in the context of this work is defined more closely to the understanding of a behavior in ethology, where it relates to the human observable actions of the robot:

Robot Behavior. *A robot behavior is the externally observable activity of a robotic system such as movement/rigor and acoustic or visual feedback produced on the robot, whether intended or unintended.*

This connotes that architectural or implementational details are not described in a pure technical manner and allows the developer and the user of a system to describe an interaction with the robot in the same way. For user-studies that shall improve the overall system from the experience of the different users, it is extremely important to be able to deduce technical aspects from the description of a *robot behavior*. A similar nomenclature for developers and testers of a system is beneficial.

[1] `http://www.ros.org/wiki/navigation`
[2] `http://www.ros.org/wiki/Robots`

A similarly ambiguous term in this context is *behavior coordination*, which is often referred to as the *action selection problem* as described by Maes [83] and specifies the selecting of the most appropriate next action at a given time. However, *behavior coordination* for a mobile interactive robot exceeds pure action selection and is defined here as follows:

Behavior Coordination. *Behavior coordination is the ability to select an action of a robotic system based on the sensing of the environment, considering interruptibility (opportunism), recoverability and interaction partners.*

This also demands a new terminology to describe the lower level parts of the system that affect the behavior, which will be introduced later in this chapter. The implementational details of the developed framework Bonsai are explained in chapter 5.

4.1. Requirements for Interactive Frameworks

On an architectural level early works such as Brooks [14] and Firby [26] have paved the way. However, for a developer of an interactive mobile robot it is important to know that the process of developing and improving the part of a software that generates a part of the *robot behavior*, these software building blocks will be referred to as *behavior modules*, is an iterative process. As it is described in Sec. 6.2, the initial implementation of such a *behavior module* often is defective or does not perform equally well in different scenarios. The idea behind this iterative process is to gradually improve the *behavior modules* over time and in different scenarios to obtain reusable building blocks that perform well in different scenarios. The following requirements, hence, help the developer during this process to improve the *behavior modules* over time. Implications of such requirements for mobile interactive robots have e.g. been discussed by Martin [84]. The following requirements are therefor also valid for other parts of the *software architecture*, e.g. the middleware, but I will focus on the impact for the framework developed during this thesis.

Modularity: For the overall system this refers to the different software components or modules of the architecture, which must be functionally independent and exchangeable. In terms of the behavior modeling this refers to the ability of putting together small building blocks to generate the *robot behavior*. These *behavior modules* also must be exchangeable and independent in terms of their execution, but may be reliant on the same resources that are also used by other *behavior modules* of *robot behavior*. This actually is also true for many software modules on a robot since they have to share the same resources as well.

Extensibility: In general this refers to the easy extension of the overall system with new software components. The extensibility for the *behavior modules* means that they too must be easily extensible with new ones (or the extension of existing ones). This especially connotes that an extension must not have side effects on other *behavior modules*.

Transparency: This is similar to the prior requirement, it means that the exchange of single *behavior modules* must be transparent to the other modules. In fact, this is also true for software components of the system. Furthermore, this means that the processing of a software component in e.g. the *functional layer* in Fig. 4.1 must be transparent for the *behavior module*.

Portability: This is a general requirement for most software components and is inherited from the requirements of high level programming to allow the usability of a software in different environments. In this case it means the usability of a *behavior module* on different robot platforms, which is only conditionally sensible in a robotics context. The implicit assumption here is of course that the platform has the same information available. A *behavior module* that is working on camera data can only be used on a platform that provides such a camera. The same is true for a *behavior module* that tracks a person, which is only possible if the platform can sense people.

Efficiency: This refers to the execution of the *behavior modules*. The framework must be able to run at an acceptable speed for interaction, which is a non-real-time system because response times can not be guaranteed. Sometimes it is also called soft real-time (e.g. in [13]), in the context of this work this only refers to the characteristics that the delay of the processing is not perceivable in a human-robot interaction.

Rapid Prototyping: This is especially important for the developers of interactive robots and refers to on the one hand easy generation of new *behavior modules* and on the other hand to the quick composition of them to allow new applications to be generated on a robot platform. The growing applicability of robots in different scenarios suggests that this should be possible even by non-programmers.

Customization: To enable the easy customization of the *robot behavior* is especially important for software intense systems and should be possible even by programmers with no detailed knowledge of the system components. With such systems it is very likely that the developer of a certain functional component (see Fig. 4.1) is not the same as the person creating the *behavior module* for a specific

application. But with growing applicability of the robot in different real world scenarios it is becoming more important that the programmer of a *behavior module* can customize this building block without detailed knowledge of the whole system (see e.g. Sec. 6.1).

Reusability: This is becoming an increasingly important aspect for robot architectures in general. From the overall system perspective it refers to easy reuse of software components on different robot systems and platforms and available frameworks. The ROS framework has done a very good job for that matter. However, in terms of the *behavior modules* it is a little different and refers to the easy reuse of the *behavior modules* for different scenarios/robots. Just like for the software components it is noteworthy that this means to use the very same *behavior modules* (e.g. classes) in different scenarios and on different robots.

Combinability: For systems with a growing number of components and functionality, coordination becomes one of the most important aspects of *robot behavior*. For developers this means that a framework should enable the effective combination of functionality and allow to exploit semantic information, e.g. of the current scene, for the coordination of *behavior modules*. It must also allow for flexible failure recovery facilitating the same information. From the system testing/evaluating perspective it must be able to change the parameters for coordination of *behavior modules*, e.g. the sequence of actions, without changing the code of the *behavior modules* itself at runtime.

Interruptibility: This is also part of the previous definition of *behavior coordination* and the interruptibility refers to the possibility of stopping a *behavior module* before it is finished while the system remains in a defined state. For interactive robots this is very important because during the execution of one task the user always should be able to change or cancel the task. For a *behavior module* this implies a decomposition of the execution into semantic subparts, more details for the implementation of this work can be found in Sec. 5.2.

Recoverability: This is also part of the previous definition of *behavior coordination* and the recoverability refers to the possibility to send a *behavior module* into a defined state even if a prior condition changes during execution. For example when the robot should fetch an object but the object is removed while the robot is already fetching it. From a software engineering perspective these requirements are related to *concurrent programming*, a good overview can be found e.g. in [10].

4.2. Bonsai Architecture: Design Principles

Apart from the terminology, there are certain design principles that have proven to not only ease the development process of interactive systems with respect to the continuous improvement over time but also help to improve the robustness of the overall system. A framework for generating *robot behavior* therefor should support these principles by providing according software structures for developers and including (runtime) reconfigurability.

In the following I will give details on the design principles that the Bonsai framework facilitates and the according design structures within the framework that construct the *robot behavior*. The focus here is to enable the efficient adaptability of the behavior and to improve the reusability of the according *behavior modules* of the *robot behavior*.

4.2.1. Separation of Concerns (SoC) & Modular Programming

SoC is the design principle of splitting up software functionality into distinct features that overlap as little as possible, as also described by Mitchell (page 5 in [89]). *Modular programming* or *modularization* as proposed by Parnas [100], is one way to achieve SoC but in terms of *robot behavior* a strict distinction of functionality in combination with hiding necessary information and a joint life cycle is equally important.

Hence, Bonsai makes use of *modular programming*, which is a well accepted software design technique, and spreads functionality of the *robot behavior* over multiple independent entities, the *behavior modules*, to encapsulate different functions of the *robot behavior* and increase reusability of the code. The code becomes more maintainable and it is a direct fulfillment of the *modularity* requirement discussed earlier in this chapter.

4.2.2. Abstraction & Readability

Kiczales [60] has introduced a model of abstraction for software engineering which outlines the general goal of abstraction: Managing complexity. This can refer to an abstraction at a level of system semantics, as e.g. shown in Fig. 4.1, but in terms of creating *robot behavior* it refers to abstraction of processes in the *functional layer* (Fig. 4.1). Developers of *robot behavior* should not be forced to cope with the full complexity of the processes underneath. As Kiczales already pointed out, the problem that may arise from this method is that even a clean interface may still sometimes carry the complexity it wants to hide into the new abstraction layer. This phenomena is sometimes also called *leaky abstraction* and emphasises that during execution the underlying complexity can not always be ignored. On a

semantic level, e.g. processes generating information, as well as on a technical level, e.g. connecting components (middleware), extending the abstraction interface with according functionality is a proposed solution. An often underestimated part of the developing process is readability of the code. The abstraction therefor should also semantically represent the processes underneath. Additionally it helps to fulfill multiple requirements such as *portability*, *transparency* and *reusability*.

4.2.3. Loose Coupling & Information Passing

Coupling of software typically refers to the degree to which each program or software component relies on one another during execution. In terms of Robot Behavior coupling can be understood as the necessity of another part of the *robot behavior* that needs to be active to be able to perform. Given the requirement for *modularity* it becomes clear that *behavior modules* need to be loosely coupled, which refers to a dependency (if any) only based on information. Typically there are two possibilities for such a loose coupling: Exchange information via parameters, e.g. calling functions, which sometimes is referred to as *data coupling* and exchanging information via sending and receiving messages of one another. This type of coupling sometimes is also referred to as *message coupling*. To reduce the inevitable overhead when creating, transmitting and translating of the information it would be advantageous to receive the according information where it is generated by the system and vice versa to send them in the same manner, as it is done via the *active memory* (see Sec. 3.3). This helps to fulfill the requirements for *extensibility* because new *behavior modules* can be more easily extended without side effects, *modularity* because every *behavior module* is a module of its own and *customization* because these modules can be easily customized for different scenarios.

Keeping these principles in mind, the main conceptual idea of the Bonsai framework is to decouple the design and decomposition of the system into reusable building blocks that can be used to flexibly construct *robot behavior*. The main goal of the Bonsai framework is to ease the developing process of *robot behavior* and produce reusable *behavior modules* by keeping the previously introduced requirements and principles in mind. The decomposition of the *behavior modules* is dependent on the contribution of the underlying processes to the overall *robot behavior* (e.g. *following a person*) rather than by its technical implementation.

In concordance with common robotic terminology the *robot behavior* modeling with Bonsai builds up on the concepts of *sensors* (see Def. 4.2.4) and *actuators* (see Def. 4.2.5). They are the main access points to the system and allow for linking of perception (e.g. people or objects) to according action of the robot (e.g. following or grasping). This approach is informed by the behavior-oriented design by Bryson [19] and allows to encapsulate rather complex perception-action-linking

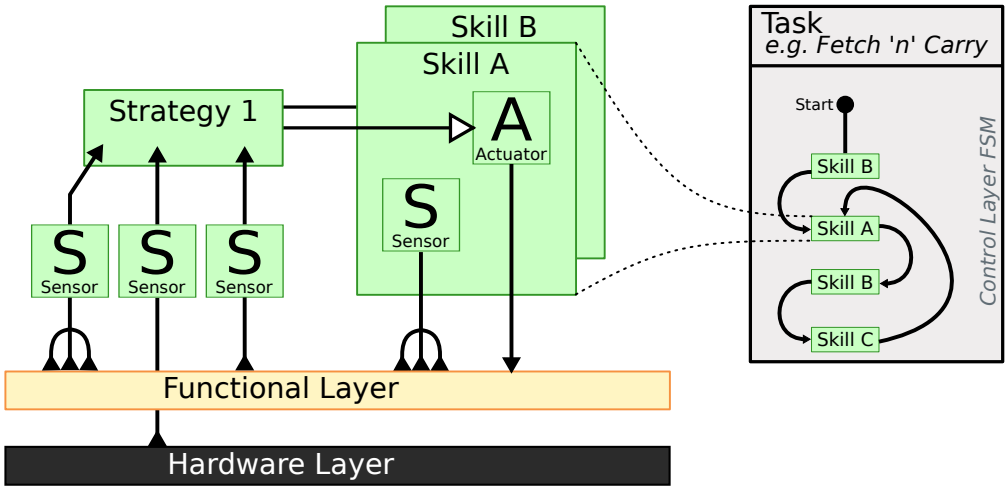


Figure 4.2.: Schema of an *Bonsai Modelling Concept* in more detail: ...

processes by only using *sensors* and *actuators*.

With these access points at hand it is possible to gather information from components or more complex processes, which are interactions between multiple components to e.g. generate a specific information, in the *functional* and *hardware layer* as depicted in Fig. 4.1 or trigger processes and actions from the same layers. The different colors from bottom to top refer to the level of abstraction from the hardware. Components in the dark gray level at the bottom either depend on direct sensory input or have a direct connection to the hardware. The obstacle avoidance module for instance depends on the input from the laser sensor at high frequencies to be able to detect obstacles while the robot is moving. The motor control represents a direct connection to the motors of the robot base to actually move the robot. Components on the light yellow level (*functional components*) in the middle depend less on the robots hardware and can process information and/or functions provided by components of the layer below or of the same layer. For example the person tracking facilitates information from the lowest layer (laser) as well as information from the face recognition. The Bonsai *sensors* and *actuators* therefor act as single well-defined synchronisation points between the system (robot platform) and the higher level *behavior coordination*.

This also means that *sensors* and *actuators* provide the interface that is used to model higher level abilities of the robot, e.g. following a person. Thereby it is possible to create reusable *behavior modules* that can be used to construct *robot behavior* to solve higher level tasks with a robot platform. This software entity of the Bonsai framework is called a *skill* (see Sec. 4.2.6).

4.2.4. Sensors: Interface for acquiring information

Interactive robots nowadays are typically equipped with a number of different hardware sensors. In contrast to the Bonsai *sensors* these are actual pieces of hardware connected to the robot that provide information about the environment. The 2011 version of the BIRON system (see Sec. 3.2) for instance has a microphone, two cameras, a 3D sensor and a laser range finder. It becomes obvious that there is additionally a lot of software necessary to generate higher level information from these inputs, e.g. there are different software components that can detect faces in a camera image and others e.g. recognize objects. On top of that another software component might again use the results of these components to generate other information, such as e.g. tracking people. All these components can also be interpreted as *sensors*. To model the *robot behavior* both types of sensors in the end provide information that needs to be available to properly model *robot behavior*.

A Sensor in Bonsai. *A sensor within the Bonsai framework is the concept of reading information from a single modularity relevant to the current system, generated by either a hardware or software component or a combination of both, to be used to model a system action and/or additional information and concealing the underlying generation and communication process.*

In Fig. 4.3 the general structure of the *Person Sensor* from the Bonsai framework is depicted. Again, the goal of this sensor is to provide information about a person in the robot's vicinity. In this example from the BIRON system, the *Person Sensor* manages the information of four components from the *functional layer* that realize the detection and tracking of person hypotheses over time as described by Jüngling [56].

4.2.5. Actuators: Interface for acting

The counterpart to *sensors* which provide information to a Please note that only some selected components of the *functional components* layer of the BIRON system (see Sec. 3.1) are shown.*robot behavior* are the Bonsai *actuators*, which similarly to sensors can represent two things: Actual hardware actuators, e.g. a robotic arm like the one on the very right in Fig. 3.4 second from bottom, or software components. *Actuators* receive information, e.g. position of an object, and then can (re)act in the real world. To model the *robot behavior*, *actuators* allow to describe what to do with which of the available information. Apart from the real hardware actuators, the Bonsai *actuators* can also be software components from the already described *functional layer*. There is a *Dialog Actuator* to trigger the dialog component of the BIRON system for example (see Peltason et al. [102]). Of

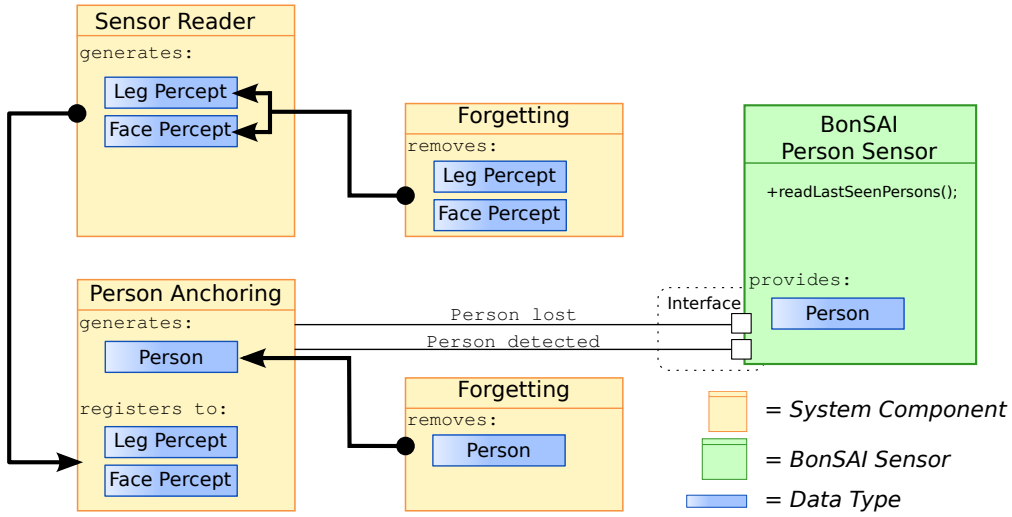


Figure 4.3.: Schema of the implementation of the *person tracking* of the BIRON system from 2009-2011.

course ultimately the speech output of the robot does require hardware (speaker) of the robot, but the actuator does neither know about that nor is it necessary to specify the exact text for the dialog.

An Actuator in Bonsai. *An actuator within the Bonsai framework is the concept of committing singular information available to the current system to a certain hard- or software component or subsystem to generate concrete action of a particular component/subsystem and concealing the underlying communication process.*

In Fig. 4.4 the general structure of the *Navigation Actuator* is described. It provides basic functions for a robot to navigate in its environment while taking care of obstacle avoidance and mapping. Despite its simple interface, the Bonsai *actuator* triggers a rather complex interaction between multiple software components from the *functional layer* which is also indicated by the color yellow for the *functional layer*. It is interesting to note that the subprocess shown is based on the 2010/2011 BIRON system which has been changed in the 2012 BIRON system without any difficulties for the *robot behaviors* modeled with Bonsai using this *actuator*.

4.2.6. Skills: Behavior Modules Facilitating Strategies

Behavior modules, as described earlier, are an important requirement to produce reusable *robot behavior* with Bonsai. Within the Bonsai framework *behavior modules* are represented via so called *skills*. The acquisition of skills in general is an

important topic in different fields of research. We can e.g. distinguish *motor skills*, which typically describe a learned sequence of movements that combined produce an action, language skills like listening and speaking or basic skills like reading and writing. What they all share is that they describe a learned capacity to carry out pre-defined actions with desired results, often optimized regarding a criteria, e.g. time or energy. Additionally, in sports studies, see e.g. in [21], a skill describes an automated stereotypical part of a behavior. Sports studies to a certain extend try to break up activities, e.g. into skills, to be able to more precisely describe the action. To describe the *robot behavior* is to some extend a similar problem because it also needs to be divided into smaller parts to be able to get a more precise description.

Guided by this understanding of a skill in general, the implementation of a *skill* within the Bonsai framework describes the main building block of *robot behavior* and is defined as follows:

A Skill in Bonsai. *A skill describes a combinable and reusable stateful building block of a* robot behavior *that covers one desired outcome (*minimal*) by only facilitating the frameworks sensors and actuators.*

As I have already explained, with Bonsai the interpretation of the terms *sensor* and *actuator* is wider and reaches beyond a simple hardware abstraction. This characteristic enables us to create different *skills* to construct more complex behavior of the robot.

From a more technical perspective a *skill* can be understood as the one place to initiate a system interaction to achieve a certain goal, e.g. to follow a person. It is stateful not because it defines a sequence of actions (*robot behavior* view) but because the interaction between the *Bonsai level* (see Fig. 4.1) with the underlying robotic system runs through different states (system view). As e.g. described by Lütkebohle [78] the interaction of software components in distributed event-based systems (DEBS) follows a certain life-cycle, making the communication between software components stateful. It can be e.g. *initiated*, *running*, *done* or *cancelled*. Guided by this understanding of the interaction between software components, the interaction between the *skills* in general are also described by a life-cycle all *skills* in Bonsai share. The phases that form the *skill life-cycle* are: Interrogation (information retrieval), activity and reclamation (information publishing).

During the *interrogation* phase a *skill* checks and retrieves all necessary data from the system components, emphasizing an important characteristic of *skills*: Since they are only constructed with the use of *sensors* and *actuators* all information and according possible actions are available in one place, even if the functionality is spread over many software components of the system. This attribute of *skills* is referred to as being *local*.

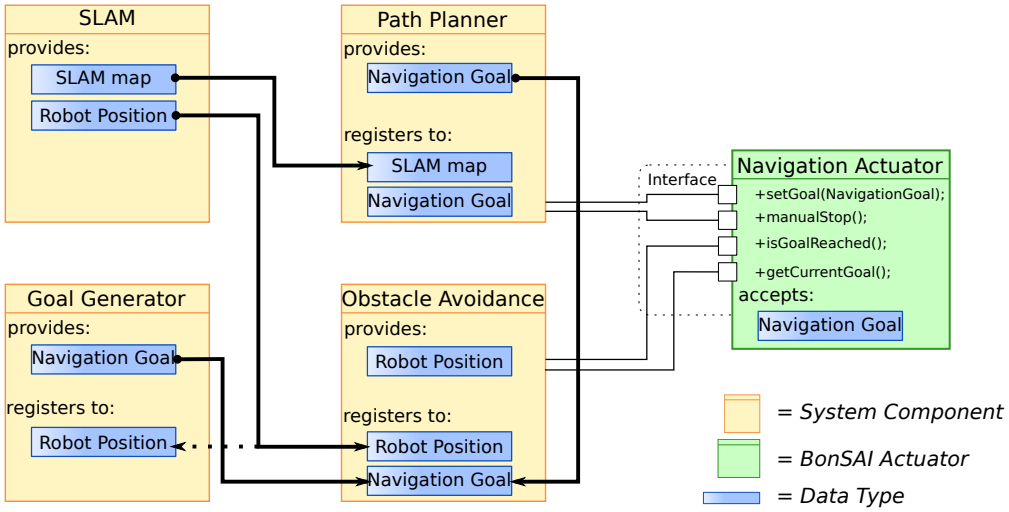


Figure 4.4.: Schema of the implementation of the *navigation* of the BIRON system of 2009-2010.

The *activity* phase describes the part of the *skill* where the action of the robot is happening. This being said it is important to note that this is often triggering and checking processes from the functional and hardware layer (see Fig. 4.1) and processing of the according information rather than containing expensive computing.

During the *reclamation* phase information that has been altered in the activity phase or newly generated information is made available to the system, by publishing it to respective *active memories* (see Sec. 3.3).

This life-cycle enables *skills* to control the robot actions more flexibly by reacting to individual results of the different phases. Hence, it is possible to provide multiple handling for error cases during certain activities of the robot inside a *skill*. A simple following *skill* for example, where the robot follows a person in a certain distance, would check if the correct person is standing in front of the robot, follows that person and additionally provides mechanisms of what to do when e.g. the guiding person was going to fast and the robot lost "visual contact".

By only relying on the *sensors* and *actuators*, the *skills* are thereby detached from the respective hard- and middleware underneath. This is in contrast to modeling the *robot behavior* inside the software components of the different layers (see Fig. 4.1).

This increases the re-usability of the *skills*, which will be explained in more detail in chapter 5. The approach of e.g. Nesnas [96] was to developed a domain-specific robotic architecture that focuses on the reduction of the overhead of developing custom robotic infrastructure, e.g. middleware, and on integrating components

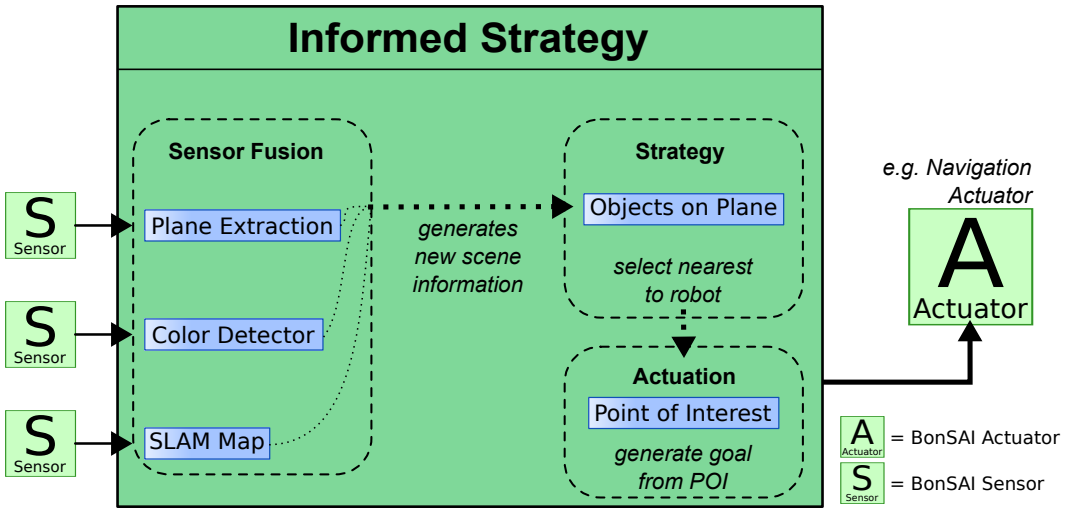


Figure 4.5.: Schema of an *informed strategy* in more detail: The *sensor fusion* on the left generates information for the *strategy*. The *Actuation* generates e.g. a goal to which the robot can navigate.

from various domains that can be reused on different hardware platforms introducing two levels of software components: The functional layer and the decision layer, which comprehend certain properties that structure the architecture and ease up the interaction between components. In contrast Bonsai focuses on explicit modeling of the *robot behavior* via *behavior modules* and their re-usability in different scenarios.

This means that *skills* model the higher level functionality of a robotic platform that uses the information generated on the levels underneath, e.g. information of the person tracker or the object recognition as shown in Fig. 4.1. Their combination can be used to solve different tasks (see Fig. 4.2, Control Level FSM on the right) that are required in different scenarios. To be able to combine *skills* in such a way the definition requires *skills* to be *minimal*, which means that each *skill* should only cover one particular part of a behavior (e.g. following a person). However, for this one aspect of a robot behavior a *skill* may cover many cases (e.g. error recovery) the robot may encounter.

The ability to include different mechanisms to handle problem situations inside a *skill* exposes the need for a structure to also describe these mechanisms in an appropriate way. This is especially important when dealing with reusable building blocks because one mechanism to handle for example the situation of loosing visual contact with a person while following might be applicable in other *skills* as well.

One outcome of the *minimal* characteristics of *skills* is that, during the *activity phase*, there are very few (often only one) *actuators* involved to describe what

the robot should do. Using the person following task again as example, the *skill* primarily needs to make the robot move, which means that the *Navigation Actuator* is used. Hence, the *activity phase* of the person following mainly configures this *actuator* to figure out where the robot needs to go next. The entity to model this modality within Bonsai is called a *strategy* and can be defined as follows:

A Strategy in Bonsai. *A strategy within Bonsai describes the modality of parameterizing a single actuator of the Bonsai framework by using information of possibly multiple sensors.*

Strategies are used to react to unexpected situations during the *activity phase* and they determine the way an *actuator* is controlled in Bonsai (changing parameters). The application of a *strategy* is derived from the *actuator* it is used for, whereas the semantics is defined by the sensors it uses and the processing that is necessary for the sensor information. Sticking with the following example, a simple *strategy* can determine e.g. the distance the robot should keep when following a person by using the person sensor and generating a target for the *Navigation Actuator*, keeping a certain distance from the person. More complex *strategies* can be distinguished by the processing steps required to control an *actuator*. This is important in situations where the selection of parameters or goals for a *skill* is not as simple, e.g. where to search for objects. However, with the different *functional components* at hand it is often possible to enable a smart selection by fusing multiple information from different components. To be able to benefit from this fusion the *informed strategies* where introduced in Bonsai. They have an additional processing step to combine multiple sensor information (sensor fusion) which generates additional information that is used to guide the *robot behavior*. This can be used to gain a better scene understanding and exploit that to enhance or optimize the *robot behavior*.

One situation where this approach was successfully implemented is described by Ziegler et al. [149]: A search behavior of a mobile robot that uses *plane extraction*, *color detection* and a SLAM map to generate target positions where the robot should search for certain objects. Assuming that objects are typically placed on planar surfaces it was possible to significantly reduce the search time for an object in a real world apartment. Later this approach was implemented into the Bonsai framework via the *informed strategies* as described in [119]. This anchors the capability to exploit additional scene information to the Bonsai framework which has been show to work well in real world scenarios, e.g. during the RoboCup competition (see Sec. 6.2).

In Fig. 4.5 the *informed strategy* shows a scenario in which information from a plane extraction, a color detector and a SLAM map are fused to generate target positions for the robot, in this case possible locations of an object. The *strategy* part in this illustration (top right corner) represents a *simple strategy*: From all

the points of interest the one which is nearest to the robot is selected. The final step is the actuation itself where a target position of the robot is generated that is suitable for the according actuator.

4.3. Contribution

In this chapter the different building blocks for *robot behavior* in the Bonsai framework where introduced. Before I will describe the implementation of the framework in the next Chapter, I want to sum up the advantages of the Bonsai framework and the modeling entities described here.

The introduced building blocks make modeling of *robot behavior* and adapting *skills*, even for developers who are no experts in any of the functional components, feasible. Especially for complex systems that are often shared by developers and system evaluators to improve the system performance over time this paradigm for an iterative modeling approach has proven (see Sec. 6) to work very well. As mentioned above, this also means that *strategies* are not only part of the *skills* but also important for increasing the reusability of these *behavior modules*. When system evaluators test the performance it is helpful not to spread the *behavior modules* over many components of the system. Keeping the local characteristics of *skills* helps to transfer observations from real world user interactions into the actual code that controls the robot. Thus, *skills* are independent of the way the underlaying system components are implemented and if problems are identified in user studies there is only one place where they need to be changed. Moreover, Bonsai *skills* can more easily be transferred to other platforms and architectures, which will be further explained in Sec. 6.3.1.

Another aspect of *robot behavior* implemented using *skills* is the reusability not in terms of platforms but in terms of scenarios. *Skills* need to be minimal and modular, e.g., only model one function of the robot at a time, to enable a flexible combination of many *skills* for different scenarios. These three aspects (local, minimal, and modular modeling) additionally ease an iterative design process for developers, because they can focus on designing functionality rather than the component configuration, which requires a detailed knowledge of the whole system. Additional insights gained in user studies can be implemented into individual *behavior modules*, which are then easily reusable for new scenarios or platforms. This also implies that the behavioral spectrum of the robot can easily be extended, new or even experimental *skills* can be combined with proven and tested *skills* in a scenario with low effort.

The two main challenges for mobile interactive robots that have been tackled here are how to integrate findings from user studies easily and efficiently and how to enable an easy reuse of *behavior modules* that have been evaluated in other

scenarios. These challenges can only be met with a framework that abstracts from the multitude of components within the system and allows changes to be made in only one place not requiring a deep knowledge about the whole system. In fact many of the concepts for the Bonsai framework were inspired by concepts that are also used in user studies because only in such studies it is possible to determine concrete requirements for future system design. In this regard the concepts presented here provide the necessary level of abstraction and foster an iterative design cycle for *behavior modules*.

Additionally, the concept of *behavior modules* indicates that there is another distinct feature for robot architectures that influences the overall performance of a system: How and where the *robot behavior* is modelled and implemented.

5. Implementing the Bonsai Framework

> Any fool can write code that a computer can understand. Good programmers write code that humans can understand.
>
> Martin Fowler

After we have learned about the architectural concepts that abet the reusability of the *behavior modules* composing the *robot behavior*, I will now give a closer look into the implementation of the Bonsai framework. Following the definition of a framework by Johnson [55], which states that a set of classes that embody an abstract design for solutions to a family of related problems and supports reuses at a larger granularity than classes can be considered a framework, *Bonsai* can be described as a domain-specific behavior-orientated Java framework.

The growing functionality of robotic systems today is accompanied with an increasing number of different software components that enable the robots to properly act in e.g. domestic environments. Unfortunately this also means that the complexity of the system increases when trying to produce more complex *robot behavior*.

There has been (and still is) a lot of research to help developers with the resulting challenge of integrating many different software components into a system and it has produced a number of robotic frameworks to actually assist the developers to put together a system of components and provide middleware functionality. Popular examples are the Player/Stage project [38], the YARP framework [88], the XCF framework [144] and the Robot Operating System (ROS) [109].

With these frameworks at hand it is possible to deploy an autonomous system in different scenarios and applications but the available frameworks also indicates that neither an existing modeling approach nor available framework covers all aspects necessary in robotic applications yet. The resulting variety of frameworks and approaches poses an additional challenge for developers of interactive robotic systems: How to model, implement and improve *robot behavior* that can be reused in other scenarios, on other platforms or with different frameworks underneath? As explained earlier todays robotic systems confront developers with a lot of in-

formation from various sources and various research areas that might be helpful for solving a higher level task with such a system.

We can conclude that the overall complexity of *robot behaviors* rises with the complexity of the system, especially in an environment where the robot interacts with a human partner. A developer must be able to focus on the relevant aspects of the *robot behavior* and should be able to combine the different information in a simple and reusable manner. It is important to ease the development process of *robot behaviors* and provide tools to combine the different information sources and *behavior modules* because the environment and the available capabilities of robot platforms have become more complex.

In contrast to modeling the *robot behavior* with different classes of messages on top of a communication layer, hence, focusing on the software components and their interaction as it was described by Lin [71] I have taken a slightly different perspective on this topic for the implementation of the Bonsai framework. Based on the concepts described in Sec. 4 Bonsai aims to model the *robot behavior* using *skills* rather than the component interaction and thereby elicit the responsible code from the *Functional Components* (see Fig. 4.1) into its own layer, referred to as *Bonsai* or Behavior layer. The guiding question during the development of Bonsai was:

How to model, implement and improve robot behavior that can be reused in other scenarios, on other platforms or with different frameworks underneath?

What this question implies is how to model the *robot behavior* without including middleware or scenario specific code. In this regard the framework enables component developers to focus on the functionality by following a simple process informed by the Information-Driven-Integration approach [144] (see Sec. 6.3) whereas developers of *robot behaviors* can reuse existing *skills* and focus on the task the robot should solve, namely the scenario (see Sec. 6.2).

To give a better insight into how the different software entities are implemented in the Bonsai framework I will first introduce the *sensors* and *actuators* here and the according communication framework. This is followed by the handling of information exchange inside Bonsai. The last part of this chapter will describe the control flow aspects and reusability of *skills* using a statechart-based implementation as *behavior engine* that allows to easily combine different *skills* in different scenarios.

5.1. Communication: Abstraction for Consistency

To achieve this decoupling of the behavior-relevant code from the middleware underneath it is very important for developers to establish a general interface that allows on the one hand to exchange necessary data from the system and on the

other hand provide an easy and ceaseless interface. Thus the *sensor* and the *actuator* interface detaches the behavior code, that only facilitates these interfaces, from the system's software components and the system's middleware by implementing a factory design pattern (see in [33]) for *sensors* and *actuators* for a specific configuration. This also allows to have middleware-specific implementations of *sensors* and *actuators* for different middleware or different systems while leaving the *skills* and their code untouched.

This also emphasizes the requirements of the system components, as introduced in Sec. 4.1, since an interface like this can only operate properly with the system components providing all necessary information, e.g by making them available in a central memory or by publishing them via publish/subscribe methods. As far as modeling *robot behavior* is concerned, we can distinguish two main types of an interface for this matter: One that does provide information (e.g. a listener or subscriber) used by the according *skills*, e.g. information coming from a camera or a laser range finder, and the other one receiving information to trigger action (e.g. a caller or Remote Method Invocation (RMI)), e.g. information to operate a robot arm or to move the robot base. A similar concept to construct system components facilitating so called *information sinks* and *information sources* is described by Lütkebohle [80]. This is also reflected by the fact that modern middleware solutions do provide implementations for at least one of these communication patterns. An overview can be found in [90] and [138]).

5.1.1. Sensors and Actuators

The basic interfaces for *sensors* and *actuators* are contained in the Bonsai core. Since the semantics of a *sensor* mainly is defined by the data content, it is possible to define common functions that apply to all *sensors* (see Listing 5.1), which basically allow to access the information of the *sensor* via the `readLast()` function in line 4 and `readLast(long timeout)` in line 5.

```
1 ...
2 public interface Sensor<T> extends ManagedCoreObject {
3    Class<T> getDataType();
4    T readLast() throws IOException, InterruptedException;
5    T readLast(long timeout) throws IOException,
        InterruptedException;
6    boolean hasNext();
7    void addSensorListener(SensorListener<T> listener);
8    void removeSensorListener(SensorListener<T> listener);
9 }
```

Listing 5.1: The simple *sensor* interface from the Bonsai core.

For *actuators* it is a little different, since the usage and the designated action of the *actuator* mainly defines the semantics, which makes it difficult to have one

single *actuator* interface for all use cases. Therefore, the *navigation actuator* serves again as example as a common *actuator* (see Listing 5.2) that is needed for any mobile platform. The core interfaces are generated by the factory implementation, which allows to define object instances of *sensors* and *actuators* that contain a certain set of functions without specifying the exact class instance of the object.

Additionally this helps to avoid duplicate code, since the creation of a *sensor* or *actuator* for a specific middleware normally differs little. For the developer of a system behavior the consistency of the functions and the availability of information is the most important aspect, which is well covered by this pattern.

```
1 ...
2 public interface NavigationActuator extends Actuator {
3     void setGoal(NavigationGoalData data) throws IOException;
4     GlobalPlan tryGoal(NavigationGoalData data) throws IOException
          ;
5     void drive(double distance, LengthUnit unit) throws
          IOException;
6     void turn(double angle, AngleUnit unit) throws IOException;
7     GoalReachedData driveTurnDone() throws IOException;
8     GoalReachedData isGoalReached() throws IOException;
9     void manualStop() throws IOException;
10    NavigationGoalData getCurrentGoal() throws IOException;
11 }
```

Listing 5.2: A stripped down version of the NavigationActuator interface from the Bonsai core.

Sensors and *actuators* extend the `ManagedCoreObject`, which is produced by the `CoreObjectFactory` within the Bonsai core. As described in Fig. 5.1 these *CoreObjects* provide a monitoring functionality, e.g., the connectivity of *sensors* and *actuators* with the system underneath, that may not be available from the underlaying middleware but still is important for the robot behavior.

For example the XCF framework [1] (see Sec. 3.3), which was mostly used on the BIRON robot, does not provide an auto-reconnect function when e.g. a functional component crashes. The `CoreObject` allows to model this functionality without interfering with the actual function of the *sensor* or *actuator*.

The implementations for a specific middleware of each *sensor* and *actuator* are contained in a separate package that uses the Bonsai core. Even though the XCF framework was most often used there have been experimental implementations for YARP [2], ROS [3] and most recently the implementation of all *sensors* and *actuators* is ported to the RSB [4] framework.

[1] `http://xcf.sourceforge.net`
[2] `http://eris.liralab.it/yarp/`
[3] `http://www.ros.org/`
[4] `http://code.cor-lab.org/projects/rsb`

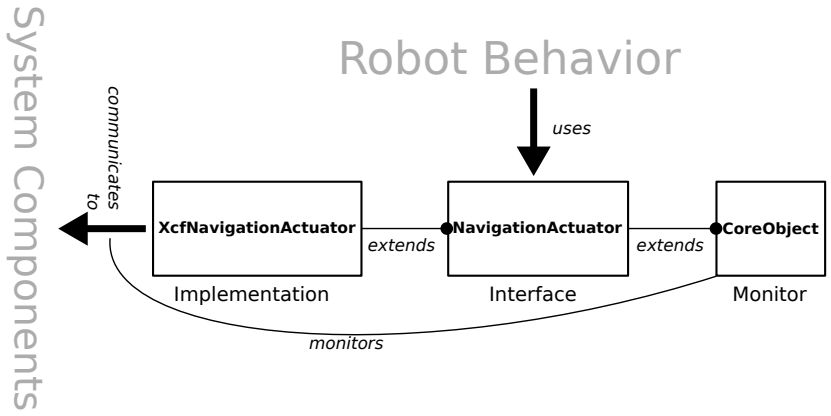


Figure 5.1.: Schematic view of the use and implementation of *sensors/actuators* in Bonsai with the *NavigationActuator* as example.

Sticking with the `NavigationActuator` as example, the actual implementation for the XCF framework is provided by the `XcfNavigationActuator` class. As depicted in Fig. 5.1 the Bonsai *skills* only know about the *sensor/actuator* interface (`NavigationActuator`), whereas the `XcfNavigationActuator` class communicates with the system components using a system specific middleware (in this case XCF) and the monitor (`CoreObject`) is checking the communication and can e.g. initialize a re-connection to the *functional component*.

Going back to the *Requirements* discussed in Chapter 4.1 this structure fortifies Loose Coupling and allows developers to focus on either the development of the *robot behavior* or the system component. This complies with the concepts, as introduced in Sec. 4.2, of Separation of Concerns (SoC) as well as Abstraction & Readability since the developer can write *skills* independently of the middleware or platform. Thus, the reusability and portability are increased and developers can more easily generate new *skills*, as e.g. required for rapid prototyping.

5.1.2. Data Content: Necessity for decoupling

One aspect of the implementation that was already mentioned before is the information exchange within the Bonsai framework. For the *sensor* interface of the Bonsai framework this is highly important, because the unitary interface design implies an additional challenge for the developers of a *robot behavior*: What information is exchanged between the system components and the Bonsai layer (see Fig. 4.1) and between the *skills*? Even more important for the developers of *skills* is what minimal set of information is necessary for these interfaces to work as intended? As mentioned earlier a *skill*, e.g. following a person, can only operate

properly if the according information about a person is available. This is a source to undercut the decoupling of the *skills* from the middleware or system underneath because it can introduce middleware or system specific dependencies. To avoid this, Bonsai needed its own way of handling information inside the framework.

To allow Bonsai *skills* to be actually detached from the specific middleware of the system the *Bielefeld Type Library (BTL)* was introduced to deal with data representation within Bonsai. In fact, the BTL is strongly influenced by the message oriented data exchange of the different systems in the Bielefeld research environment.

The BTL was implemented very early in the development process of Bonsai and mainly provides parsing functionality for XML-based messages exchanged between the system and the Bonsai Layer and among the *skills*. The main benefit of using XML-based representations, apart from the human readability of the exchanged information, is the extensibility which allows to have minimal information requirements for e.g. *sensors*, which can be extended for new *sensors* or *actuators* without the need to change the existing ones.

```
<LIST type="ANNOTATION">
  <ELEMENT_TYPE>de.unibi.airobots.btl.data.map.Annotation</
     ELEMENT_TYPE>
  <ANNOTATION label="kitchen">
    <VIEWPOINT category="VIEW" label="fridge">
      <ROBOTPOSITION>
        <POSITION kind="absolute" ref="world" theta="0" x="4.75"
      y="29.95000076293945"/>
      </ROBOTPOSITION>
    </VIEWPOINT>
    <VIEWPOINT category="VIEW" label="kitchen_table">
      <ROBOTPOSITION>
        <POSITION kind="absolute" ref="world"
      theta="1.700000047683716" x="3.700000047683716"
      y="30.29999923706055"/>
      </ROBOTPOSITION>
    </VIEWPOINT>
    <PRECISEPOLYGON>
      <POINT2D scope="GLOBAL" x="2250" y="32150.001953125"/>
      <POINT2D scope="GLOBAL" x="2200" y="28400"/>
      <POINT2D scope="GLOBAL" x="5450" y="28200"/>
      <POINT2D scope="GLOBAL" x="5500" y="32150.001953125"/>
    </PRECISEPOLYGON>
  </ANNOTATION>
</LIST>
```

Listing 5.3: Short example for a *MapAnnotation* XML, describing the *kitchen* area (precisepolygon) and a *view point* for the fridge.

This allows to e.g. define data types that are necessary for *skills*, such as a

NavigationGoal which describes a target position of the robot on a map, which do not have to have a correspondent data type on the functional layer of the system. But at the same time this allows all *sensors*, *actuators*, *strategies* and *skills* to work on the same set of information. Another example is the *Person* type of the BTL. It contains information about the angle and the distance of a person as seen from the robot position (ego view), but additionally may hold information if a *Person* is the current interaction partner of the robot and the name as well as additional percepts (face, pair of legs) of the person. It is getting clear that the information that compose a *Person* can vary between different systems. In fact this information can even vary between different software components of the same system.

A practical example for the usability of the BTL types are the Strategies. They produce a single output that can be used by an *actuator*, e.g. a *NavigationGoal* to be used by the *NavigationActuator*. This increases the reusability and eases the process for the developer since one has to only deal with these kind of data types. Additionally the BTL provides unit support and convert functionality for convenience of the developer and to make sure that e.g. distances are always the same as well as angles and can be easily converted if necessary.

Listing 5.3 shows a stripped-down version of a *MapAnnotation* which is used to represent information about the surrounding of the robot based on a pre-recorded SLAM map. The example shows the annotation for the label *kitchen* (see line 3) and the area on the map covered by this label (see lines 17-23). The *viewpoint* labeled *fridge* (see line 4) contains a *robot position* (see lines 5-8) that describes a position on the map from which the robot can perceive an object or area. In this case it can see the fridge, multiple *robot positions* for one label are possible.

5.2. Control Flow: Enabling Reusable Building-blocks

The most important part of the Bonsai framework is the implementation of the *behavior modules*. Not only because they in the end generate the *robot behavior* but also because the requirements presented in Sec. 4.1 have a big impact on *behavior modules*: They need to fulfill all of them and are influenced by the *sensors* and *actuators* as well as the control layer on top. This means that to meet the requirements it is important to restructure the way the functionality of a system is implemented. This will in the end help the developer to easily implement *behavior modules*, thus, increasing the behavioral spectrum of the robot.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0"
    initial="ready">
```

```
3       <state id="ready">
4           <transition event="watch.start" target="running"/>
5       </state>
6       <state id="running">
7           <transition event="watch.split" target="paused"/>
8           <transition event="watch.stop" target="stopped"/>
9       </state>
10      <state id="paused">
11          <transition event="watch.unsplit" target="running"/>
12          <transition event="watch.stop" target="stopped"/>
13      </state>
14      <state id="stopped">
15          <transition event="watch.reset" target="ready"/>
16      </state>
17  </scxml>
```

Listing 5.4: SCXML example of a stop watch with states *ready*, *running*, *paused* and *stopped*.

As it was explainer earlier in Sec. 4.2.6 the Bonsai *skills* are the implementation of *behavior modules* that is used in the Control Layer (see Fig. 4.1) to generate a *robot behavior*. In this Section I will give a detailed introduction to the Bonsai *skills*, how they are structured and why and how they are combined and executed. The implementation of the control layer in Bonsai relies on statecharts (see Exc. 5.2.1) and the according working draft published by the World Wide Web Consortium (W3C) for SCXML [5]. There are different implementations available for SCXML, the one used in Bonsai is the Apache Commons SCXML library [6]. This is because it is not only capable of parsing and validating SCXML but also provides an execution environment for the resulting state machines and because it was published under an open source license it is reasonably easy to extend and maintain. A short example for a simple stop watch defined in SCXML can be found in Listing 5.4. It describes a state machine that has four states, *stopped*, *paused*, *running* and *ready* with *ready* being the initial state. The transitions are defined as events, e.g. *watch.start*, and each have a target state, referred to by name.

5.2.1. Skills

All Bonsai *skills* extend the *AbstractSkill* class, which basically defines the life cycle of any *skill* within the Bonsai framework (see Fig. 5.4), a stripped version of the class can be found in Listing 5.5. As it was explained in Section 4.2.6 a *behavior module* can be split into three phases: Interrogation, activity and recla-

[5] http://www.w3.org/TR/scxml/
[6] http://commons.apache.org/scxml/

Statecharts

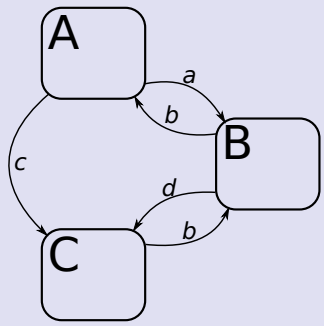


Figure 5.2.: Simple state diagram.

The so called *state and transition explosion*, sometimes also referred to as exponential blow-up, describes the exponential growing multitude of states and according transitions when describing a complex system with simple state diagrams (as seen in Fig. 5.2). They require a distinct node for every valid combination of parameters that defines the state. However, statecharts are a visual formalism [46], which is also part of the Unified Modeling Language (UML), to tackle this problem and provide modular, hierarchical and well-structured state graphs for complex systems by allowing the modeling of superstates, orthogonal regions, and activities as part of a state.

With statecharts it becomes possible to model multiple state diagrams within one statechart, which allows to have transitions between internal (sub-)state machines without affecting one another in the statechart. This circumvents the exponential blow-up and also introduced the *history* of states. This allows to model complex systems more easily, because it is possible to enter a group of states (superstate) by referring to the *history*, which typically refers to the state within that superstate that was last active.

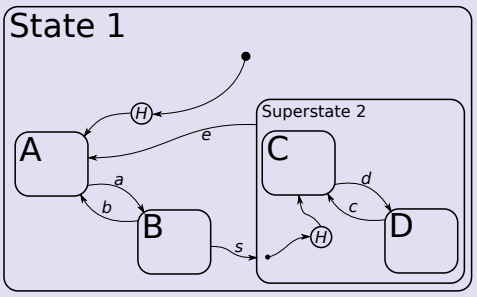


Figure 5.3.: A simple statechart.

More recently the development of the *State Chart extensible Markup Language* (SCXML) [a], which provides a generic state-machine based execution environment, has gained attention by different communities as a standard implementation for state machines.

[a] `http://www.w3.org/TR/scxml/`

Excerpt 5.2.1: Statecharts: A Visual Formalism

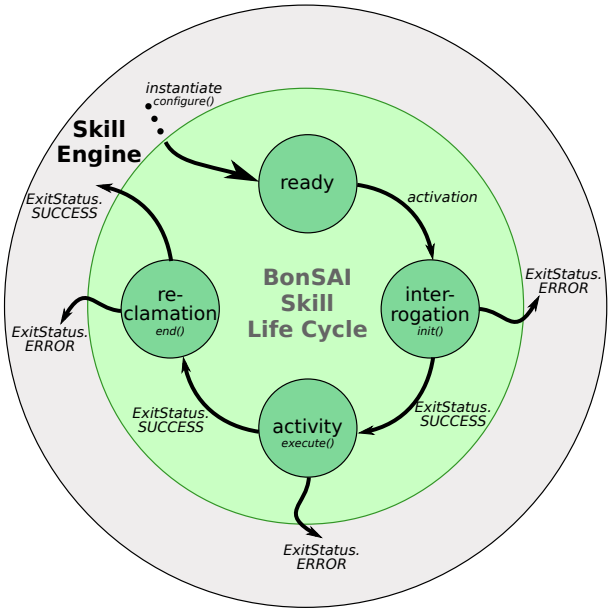


Figure 5.4.: The life cycle of a Bonsai *skill*.

mation. Theses phases are represented by the abstract functions described in the *AbstractSkill* in lines 12-14, which means the developer can implement the functions according to the needs of a *skill*, thus, implicitly abiding the requirements.

The *AbstractSkill* also provides status information that are used to communicate the success of each phase to the *skill engine* as well as the result of the *skill* itself. They are described in Listing 5.5 (lines 6-8). Obviously *SUCCESS* represents a successful execution of a phase, whereas *ERROR* stands for a precondition that does not hold for the next phase. If, for example, certain information that are required by the *activity* phase are unavailable during the `init()`, then the *ExitStatus* will be *ERROR*. If there is a software failure during one of the phases, e.g. a *sensor* is unreachable (failed connection), the *ExitStatus* will be *FATAL*. This distinction enables the developer to react to semantic failures, which are normally recoverable from within the *skills*, whereas a software failure normally is not recoverable from within the *skills*.

The *skill engine* describes the execution environment for all Bonsai *skills* based on an SCXML state machine. For execution of the state machine as well as parsing/validating the SCXML description the Apache Commons library, as mentioned above, was used and extended. The engine also allows to communicate the *ExitStatus* of each phase facilitating *events*, which means that the `init()`, `execute()` and `end()` functions of each *skill* will always trigger such an event. In turn the *skill engine* holds a state machine object as well as a system configuration to enable

the activation of the different *skills* as well as canceling *skills* when the situation (or interaction) has changed.

```
...
public abstract class AbstractSkill extends Thread {

  public enum ExitStatus {

    SUCCESS("success"),
    ERROR("error"),
    FATAL("fatal");
    ...
  }
...
  public abstract ExitStatus init();
  public abstract ExitStatus execute();
  public abstract ExitStatus end();
...
}
```

Listing 5.5: Stripped down Java class for an AbstractSkill.

The *skill life-cycle* is depicted in Fig. 5.4 combined with the interactions with the *skill engine* between the different phases. On robot startup a Bonsai instance, called the *BonsaiManager*, creates the different *sensors* and *actuators* according to a system configuration file (see App. A.2). After that the *skill engine* instantiates all necessary *skills* as described in the SCXML file, facilitating the *sensors* and *actuators* of the *BonsaiManager* via the *configure()* function depicted in the upper left of Fig. 5.4. When ready a *skill* is activated by the *skill engine* and enters the interrogation phase by calling the `init()` function of a *skill*. If not successful (ERROR or FATAL) an event is triggered to inform the *skill engine* and the *skill* itself is stopped (see right on Fig. 5.4). Otherwise the activity phase is entered by calling the `execute()` function which will produce an action of the robot. If successful the last phase, reclamation phase, is entered by calling the `end()` function. Additionally the *skill engine* can cancel a *skill* during the phase transitions, e.g. when a parallel *skill* has detected a change of the current situation or when a human user gives a new command to the robot.

Which *skills* are activated and when is defined by an SCXML which eventually is describing the *robot behavior*. This state machine also holds the information on how to react to the triggered *ExitStatus* events, e.g. what to do when necessary information is not available.

5.2.2. Control Layer and SCXML

The Listing 5.6 shows a shortened SCXML file describing the activation of the different Bonsai *skills*, in this case a *skill* setting a target position according to

a map annotation (*SetTargetByAnnotation*) followed by a *DriveToPosition skill* that will check if a position was set (Interrogation) and than try to reach the position with the robot. A complete example for an SCXML used with Bonsai can be found under A.3, the basic core features of the current SCXML standard can be found online [7].

```
<scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0"
initial="navigation.SetTargetByAnnotation">
    <state id="navigation.SetTargetByAnnotation">
        <transition event="SetTargetByAnnotation.success"
    target="navigation.DriveToPosition"/>
        <transition event="SetTargetByAnnotation.error"
    target="Fatal"/>
        <transition event="SetTargetByAnnotation.fatal"
    target="Fatal"/>
    </state>
    <state id="navigation.DriveToPosition">
        <transition event="DriveToPosition.success"
    target="End"/>
        <transition event="DriveToPosition.error"
    target="Fatal"/>
        <transition event="DriveToPosition.fatal"
    target="Fatal"/>
    </state>
    <state id="Fatal">
        <transition event="Error.restart"
    target="navigation.SetTargetByAnnotation"/>
    </state>
    <final id="End"/>
</scxml>
```

Listing 5.6: Stripped down SCXML example for setting a target position from an annotation on a map.

Two other main features of the SCXML implementation shall be introduced here since they underline the benefit of using this standard and additionally play an important role when coordinating the Bonsai *skills*: The *send* and the *parallel* feature.

As it was explained before, the *skill engine* exchanges information about the *ExitStatus* during the different phases via events which are also included in the SCXML describing the transitions between different *skills*. These events are *skill* specific because they have to be defined by the developer of the *skill*. However, when using the *skills* in different scenarios it is important to be able to add scenario specific events. For this the *send* feature of SCXML was used, as described in Listing 5.7, which allows to define individual events inside the SCXML file that

[7] http://www.w3.org/TR/scxml/#Basic

also can trigger transitions between *skills*.

```
1 <state id="Skill1">
2 <onentry>
3   <send event="'Skill1.timeout'" delay="'30s'" />
4 </onentry>
5 ...
6 <transition event="Skill1.timeout" target="AnotherSkill" />
7 </state>
```

Listing 5.7: Example for a scenario specific event handled via the Bonsai *skill engine*.

The example shows a timeout for a *skill*. The lines 2-4 define the event *Skill1.timeout* which automatically will be triggered by the *skill engine* with a delay of 30 seconds after activation of the *skill*. The transition defined in line 6 will be performed, switching to *AnotherSkill*, after 30 seconds, fully independent of the *ExitStatus* of the *skill* itself.

However, the *send* feature alone is not enough to cope with the complexity of some *skills*. Even though they should be as *minimal* as possible, some of them may still need multiple outcomes after the activity phase. A good example is the *Grasp skill* described in Listing 5.8. To achieve this all Bonsai *skills* can set a *ProcessingStatus* at the end of the activity phase which is automatically added to the *ExitStatus* event. The benefit of that is, even if the SCXML does not react to the specific event including the *ProcessingStatus*, it can still be recognised as a normal *ExitStatus* event. In the example the Grasp *skill* is able to differentiate 3 error situations and adds an according *ProcessingStatus* to each event.

```
1 <state id="Grasp">
2   <transition event="Grasp.success" target="dialog.Talk#IGotIt" />
3   <transition event="Grasp.error.noObjects" target="arm.
       MoveGripperUp" />
4   <transition event="Grasp.error.crash" target="arm.UnblockArm" />
5   <transition event="Grasp.error.collision" target="dialog.Talk#
       TryAgain" />
6   <transition event="Grasp.error" target="arm.GoUpGripper" />
7   <transition event="Grasp.fatal" target="dialog.Talk#Fatal" />
8 </state>
```

Listing 5.8: Stripped example of a *GraspObject skill* facilitating the *ProcessingStatus* for transition.

They can be identified in the SCXML and activate *skills* to e.g. cope with the situation or make the robot ask for help: No object was grasped (line 3), at least one motor of the robot arm is blocked (line 4) or a potential collision was detected (line 5). In any other case a simple error event is send (line 6).

The *ProcessingStatus* allows the developer of a *skill* to add additional information about the result of the activity phase to the *ExitStatus*. This is very helpful to differentiate error situations as described but also for *skills* that have different results, e.g. when recognizing objects or persons. The *ProcessingStatus* can also be used to prevent *skills* that are more time consuming or *skills* that need to wait for longer computations to finish (e.g. learning objects or faces) to remain interruptible. In such a situation the *ProcessingStatus* can be used to indicate that the activity phase is not finished yet and make the *skill engine* activate the very same *skill* by adding itself as a transition target in the SCXML. The same *skill* will be activated again and check if e.g. the processing is done, very much like a monitor state.

```
<parallel id="ParallelID">
  <state id="Skill1"/>
    <transition event="Skill1.success" target="SkillOne"/>
      <transition event="Skill1.error" target="ErrorSkill"/>
    <transition event="Skill1.fatal" target="FatalSkill"/>
  </state>
  <state id="Skill2"/>
    <transition event="Skill2.success" target="SkillTwo"/>
    <transition event="Skill2.error" target="ErrorSkill"/>
    <transition event="Skill2.fatal" target="FatalSkill"/>
  </state>
</parallel>
```

Listing 5.9: A shortened SCXML example for a parallel *skill*.

The *parallel* feature, which is described in Listing 5.9, allows to activate multiple *skills* in parallel. The Listing describes a situation where two *skills*, *Skill1* and *Skill2*, are activated in parallel and depending on which *skill* finishes (SUCCESS) first, the according transition is made (*SkillOne* or *SkillTwo*). Another common use case for parallel *skills* is the responsiveness of the robot. In this situation one *skill* is active, e.g. a person follow *skill*, and another *skill* is checking for user input, e.g. a stop command.

5.3. Contribution

With the implementation of the Bonsai framework it is possible to quickly put together interactive *robot behavior* for different scenarios and systems. With the Bonsai *skills* it is possible to easily reuse tested *behavior modules* to compose new *robot behavior* in new scenarios as well as test new, possibly experimental, *skills* together with proven and tested *skills* of the robot, which is very important for e.g. research platforms. This is because the same *skills* are instantiated by the *skill engine* for all different scenarios, improvements are made in one place (the

code of the skill). Due to the joint *skill life-cycle* and the usage of the same *sensors/actuators*, newly implemented *skills* can be easily included into scenarios.

As a side effect the availability of *skills* also allows for improved software testing because single *skills* can be tested separately, without the need of a whole system, and often automatically inside the build pipeline, whereas writing software tests for a whole *robot behavior* still is a bigger challenge. Hence, often only few tests for the *robot behavior* are written, which can be improved with the Bonsai *skills* to ensure at least that these parts are working as they are supposed to do.

From a system perspective the Bonsai framework allows to adapt the *robot behavior* independently of the *functional components* (see Fig. 4.1) of the system and vice versa. This greatly contributes to the stability of a frequently adapted system because it avoids unintended changes on the behavior when changing a component. This also highlights the easy separation of the actual *robot behavior* from the control flow: There are two separate instances that manage what the robot does (*skills*) and when or how it should be done (*skill engine*). This strict separation also circumvents the *sequencer* problem (see Sec. 2.1.2). Here, the sequencing is explicitly modeled via the SCXML.

6. Evaluation

> Science is the systematic classification of experience.
>
> George Henry Lewes

Evaluation is essential in order to design any kind of usable systems. For interactive robots the evaluation of the system may be even more important, not only because the interplay of many components needs to be tested under realistic conditions but especially because user interaction with a system needs to be tested since this is still difficult to do in simulation.

In this part I want to present the different scenarios that where used to evaluate the Bonsai framework. Obviously the overall system behavior does play an important role because this is one aspect that I am striving to improve with Bonsai over time but it is certainly not the only aspect.

The evaluation results for the *extended home tour* scenario will be presented in Sec. 6.1 and in Sec. 6.2 for the RoboCup competition. Additionally to this I have considered aspects that are more important for the developers of such systems: The portability of developed *skills* to other platforms/systems and the usability if the Bonsai framework itself is. Both aspects are evaluated in Sec. 6.3.

6.1. System Evaluation: The Extended Home Tour

The driving question for this evaluation of the Bonsai framework was how it could be possible to easily re-use *robot behavior* that has been evaluated in other scenarios. The hypothesis is that with the Bonsai *skills* at hand it is possible to re-use the *skills* in other scenarios than they where originally developed for and with the separate modeling of the control flow the failure cases during an interaction caused by the system rather than the *robot behavior* can be reduced. This question is highly relevant because user studies and their design take a lot of time and effort. Thus, it would be a great advantage if we were able to improve and re-use robot *skills* that have been tested and worked well in e.g. other scenarios or other studies before.

In addition to the *home tour scenario* described in Exc. 2.1.4 the scenario for this evaluation was extended: After the robot was introduced to rooms and

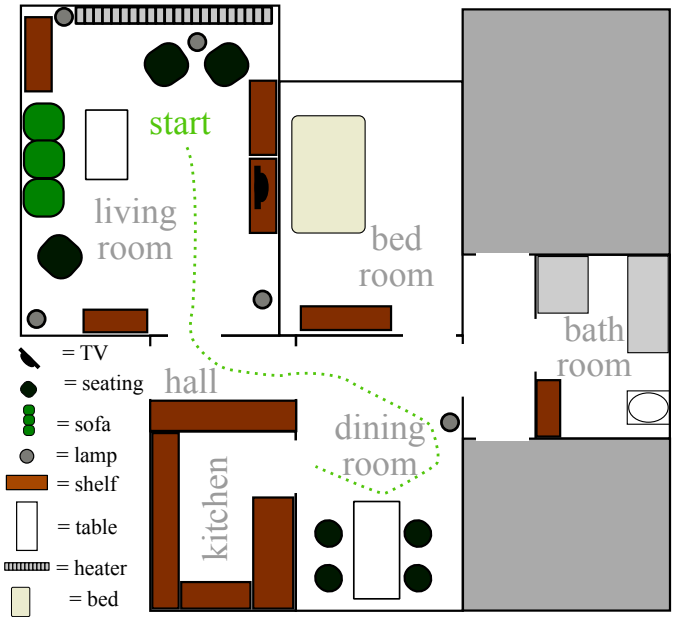


Figure 6.1.: The robot apartment. The path of the tour (green) starts in the living room via the hall to the dining room.

objects of the apartment (e.g. "This is the kitchen!"), the robot needed to guide the human to one location it has remembered before. This not only shows that the robot actually remembered what was introduced to it before but also added an autonomous part where the robot had to navigate through the apartment instead of following a person.

Participants

For this study a group of participants was invited to the robots apartment that had not interacted with the robot BIRON before. 14 participants, 7 male and 7 female, took part. Their ages ranged from 18 to 54 years with a mean of 38.9 years. Based on a questionnaire presented to all subjects prior to the study where all participants had to do a self-assessment, their experience with computers was 3.1 on a scale of 1 "*no experience at all*" to 5 "*a lot of experience*". Their experience with robots based on this assessment was 1.7 on the same scale as above (1 to 5). All participants were native speakers of German and interacted with BIRON in German. All subjects interacted voluntarily with the robot and could stop the interaction at any point without a reason.

Setup

The subjects were welcomed at the robots apartment and then received a written introduction to the study, a short description of the home tour scenario (see Exc. 2.1.4) and the robot (see Sec. 3.1) and an overview of the phases of the study. All participants had to answer a questionnaire on demographic data and their experience with computers and experience interacting with robots prior to the study. After that the subjects had a short training period on using the speech recognition of the robot (i.e. proper distance to microphone) and were asked to speak some phrases for habituation. In the following the users were handed a tutorial script for practice to reduce hesitant behaviors. In this phase the subjects also where allowed to ask questions about the robot, which was not allowed during the interaction. The script contained all commands they would need later on and the users were asked to try them out with the robot. The instruction for this main task was to:

Greet the robot,
Guide the robot from the living room to the dining room via the hall (see Fig. 6.1),
Show and label the living room and the dining room,
Show the bookshelf in the living room and the floor lamp in the dining room,
Ask the robot to go back to one of the rooms or objects and
Say good-bye.

The tasks were chosen based on the scenario of the predecessor study in 2007 and covered a number of abilities that the robot was able to autonomously perform at that point of time. The whole interaction was recorded with three cameras, one stationary camera in the kitchen, another stationary camera in the dining room and the interaction itself was recorded with a hand held camera. For the latter the camera person was instructed to stay out of sight of the subject and the robot sensors to not interfere in any way with the interaction. After the interaction, all participants were interviewed and answered a second questionnaire that included items on liking of the robot, attributions made towards the robot, and usability of the robot.

Results

There are different outcomes of this study, some are directly evident such as the logging functionality of the Bonsai *skills* that is helpful to produce reasonable data for video annotation. Another positive effect was actually having an interaction scripts in terms of a state automata on the robot (see Sec. 5.2.2) that not only helps the experimenter to determine the course of the interaction but also makes

Task	count 2010	failures 2010	count 2007	failures study 2007
Greet	14	0	45	10
Intro	14	0	5	2
Offer	133	4	–	–
Farewell	–	–	18	4
Guiding	69	12	109	9
Teaching Object	58	19	66	31
Teaching Room	47	14	40	10
Showing Object	8	2	–	–
Showing Room	6	5	–	–
Register	20	0	38	4
Obstacle	–	–	25	0
Reset	–	–	18	5
Stop	16	2	29	3
Sum	385	58	393	78

Table 6.1.: The tasks from the robot studies in 2010 & 2007. Cells marked with "–" where not part of the particular scenario.

the interaction easily reproducible for all subjects. The outcome in terms of the *robot behavior* is only apparent in contrast to the former robot study conducted in 2007 (see [76]). The analysis of the robot behavior is based on the SInA approach (Systemic Interaction Analysis in HRI [77]). In this approach the overall interaction is divided into small reoccurring interaction episodes called *tasks*. A task has a defined start and end action, e.g. the task *Greeting* starts with the user saying something like *Hello robot!* and typically ends with a confirmation of the user. An example dialog for a *Greeting* task could look like this:

human: *Hello robot!*
robot: *Hello. What is your name?*
human: *My name is Sarah.*
robot: *I understood Sarah. Is this correct?*
human: *Yes, that's correct.*

In order to analyse the interaction there where 10 different tasks in both studies which can be found in Table 6.1. Given theses tasks a simple measure for the quality of interaction is to count the overall number of tasks occurring and identify the tasks in which the interaction was unexpected or *problem related*. The expectation was to reduce the share of tasks that had problems or failures in them with the Bonsai framework, which was the case as it also can be seen in Table 6.1:

Only taking the bare numbers into account the number of problem related tasks dropped from 20% to 15% with Bonsai. Additionally the *Obstacle* and *Reset* tasks where only introduced to recover the interaction when the robot got stuck or the interaction could not be continued by the user. Especially the reset is interesting, which still would fail almost 1/3 of the time and was necessary at 18 occasions with only 14 participants. This particular problem will be discussed further in Sec. 7.2.

The RoboCup Competition

The *RoboCup* [5] (or Robot World Cup) was established in 1997 as an attempt to promote intelligent robotics research via an international scientific initiative with the goal to improve the current state-of-the-art of autonomous or intelligent robots. Since then it has become the biggest international robotics competition in the world with many different challenges for robots ranging from soccer to housekeeping. One of the main objectives reads as follows:

> By mid-21st century, a team of fully autonomous humanoid robot soccer players shall win the soccer game, comply with the official rule of the FIFA, against the winner of the most recent World Cup.[a]

Starting with robot soccer, the RoboCup today has diversified into different leagues. Every league represents a special field of robotics application, e.g. the rescue league deals with disaster management tasks where a robot has to collect information and search the environment e.g. for injured victims in cooperation with human workers. One task here is that the robot has to build up a map of the searched environment that the human workers can use to find victims. Another relatively new league is the @HOME league where robots have to autonomously fulfill tasks in a domestic home environment. One task here is e.g. to interact with people and correctly identify them to help them with house holding tasks such as cleaning. One specialty of the league is that the set of rules gets changed every 2 years to adapt to new capabilities of the robots and to provide new challenges. One future challenge is e.g. to enable robots to use public transport. Currently there are 5 main areas of research (soccer, rescue, @home, junior and logistics) organised in 12 different leagues. They all have in common that they attempt to promote research in a certain area by providing a common domain and common tasks for benchmarking.

[a] `http://www.robocup.org/about-robocup/objective/`

Excerpt 6.1.1: About the *RoboCup*

6.2. Performance & Stability: The RoboCup Experience

The RoboCup competition (see Exc. 6.1.1) provides a unique opportunity for robotic researchers to not only test their systems and methods under realistic, sometimes even harder than real-world (e.g. with an audience watching), conditions but also the chance to continuously improve system behavior over time. The guiding question in the Bonsai context was "*Can we improve the performance and stability over time by integrating experience gained from the competition into the Bonsai skills?*" and an additional question was "*Is the Bonsai framework usable even for untrained developers that never have worked with the robotic platform before*".

The RoboCup team ToBI [1] was founded in 2009 and mainly consists of students in their bachelor/master that have no experience with a robot platform like the one used in the competition (see Sec. 3.2). A long research history in human-robot interaction at the Bielefeld University clearly indicated that the composition of a *robot behavior* for such systems still is a complex task. Reusability in such systems is usually difficult because one change often needs to be made in different components of the system and the developer needs to pay close attention to dependencies and effects between components. Thus, changes take a lot of time and a lot of knowledge. As a matter of fact in the beginning of this thesis the composition of a *robot behavior* could only be done by system experts and adaptation of existing behavior was difficult and defective.

This evaluation covers the RoboCup@HOME participation of team ToBI from 2009 until 2012, which includes the RoboCup German Open as well as the World Championship of each year.

Participants

The participants where selected from the bachelor/master students at the Bielefeld University with an average age of 24 years in 2012 (2011: 24.33, 2009/10: 24.53). The programming experience was rather similar over the 4 years with a focus on Java experience as shown in Fig. 6.2. From 2009-2012 team ToBI had 34 (2009:7, 2010:8, 2011:10, 2012:9) different developers, 32 male and 2 female.

Procedure

To be able to integrate the lessons learned from the RoboCup competition in each year, this part focuses on the Bonsai relevant aspects of the preparation and the

[1] `www.cit-ec.de/ToBI`

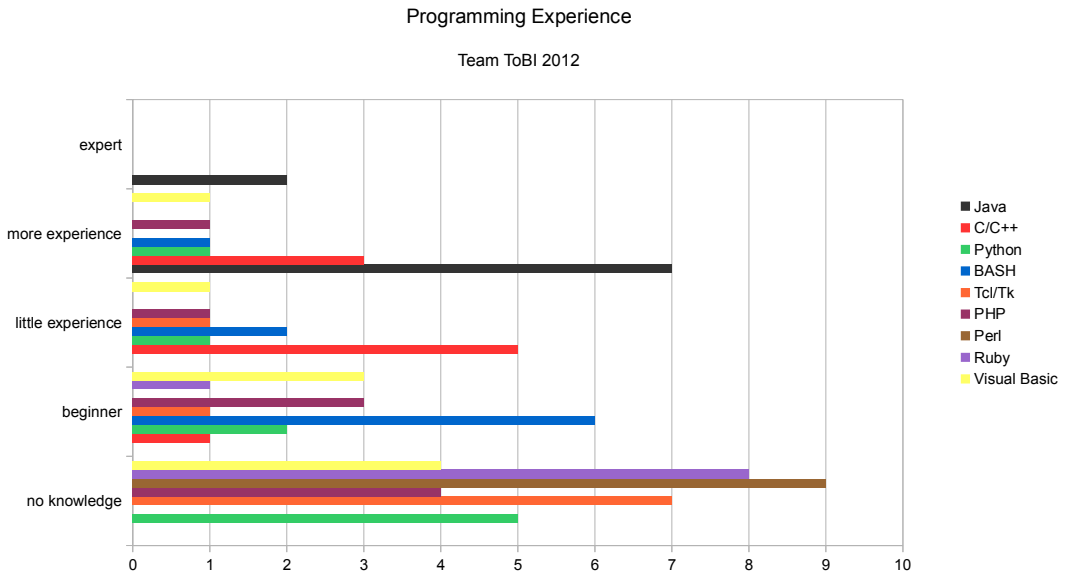


Figure 6.2.: The programming experience of the team ToBI members in 2012. An accumulated graphic for 2009/2012 as well as for 2011 can be found in B.1

competition.

Every year the developers where asked to give feedback via an online questionnaire about their experience with Bonsai (and of course the rest of the software/system). Additionally, since the evaluation is based on the competition, the reasons for e.g. tasks where we got little or no points where back-traced as well as common programming errors. This can be something like missing functionality of an actuator, e.g. the `isGoalReached()` function of the `NavigationActuator`, or repeated errors resulting from re-implementing a state machine pattern for every task. Hence, there where different main focus points over the years to tackle these problems. The driving question every year was: *Where did we loose a lot of time in the development and points in the competition?*

2009: Sensors and Actuators In the first year of the team ToBIs participation in the RoboCup@HOME it is obvious that the main functionality of the Bonsai *sensors* and *actuators* was the main focus. This directly relates to new software components that where added to the system due to tasks the robot had to perform. At the time only simple implementations of the NavigationActuator existed along with the SpeechActuator and a number rudimentary *sensors* such as the LaserSensor.

RoboCup Participants	2009/2010*	2011	2012
Average understanding Sensors/Actuators	4.00	4.11	4.44
Average Understanding of the System	3.80	3.56	3.67
Average confidence in fixing errors	4.33	4.44	4.22
Standings of Team ToBI in the World Cup	8/7	5	3

Table 6.2.: *: joint data acquisition in 2010. Data analysis of the RoboCup questionnaire until 2012. Scale: 1 *not at all* – 5 *Very good*

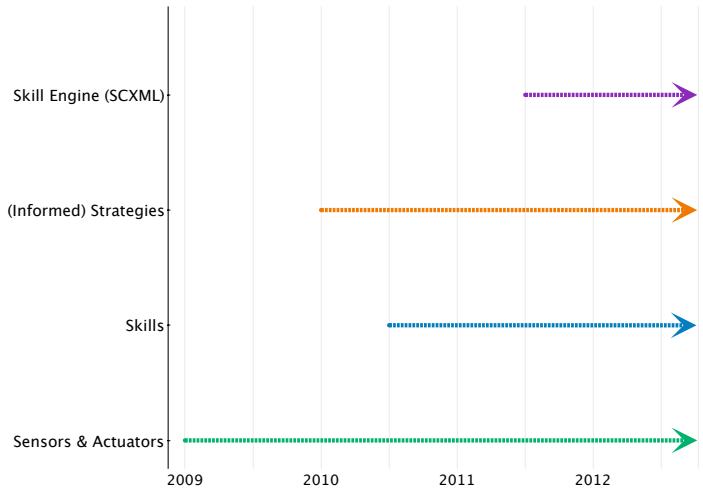


Figure 6.3.: Availability of the Bonsai features through the different years of the RoboCup competition.

The main challenge was to implement a proper `NavigationActuator` that can handle local and global navigation with a planner component. This would enable the Bonsai framework to work on a map representation. As mentioned earlier the Actuator did not support a checking if the robot reached a certain position on the map. This has lead to *robot behavior* where the robot simply stands at a position and does not proceed with necessary actions because it could not reach the desired position for different reasons (e.g. stuck on carpet/doorway). The interface was also extended to support a precision radius and angle for the robot to avoid prancing behavior of the robot. This means the robot has actually reached the goal but the low-level parameters of the navigation are more strict than necessary for the scenario. In this case the robot will try to reach a goal very precisely which leads to continuous small movements of the robot on the spot.

The second challenge within Bonsai in 2009 was the integration of person detection and recognition, which is very important for interactive robots. Within the Bonsai framework this aspect was covert with the `PersonSensor` and the `FaceIdentificationSensor`. The `PersonSensor` was first build as a simple interface to make the information of a person tracking component available to the Bonsai level. During the development it became obvious that there was more functionality needed. The person tracking component of the ToBI system only tracks persons in the robot vicinity but for the different tasks in the competition the robot needed to be able to keep information about persons outside the robots vicinity.

The `FaceIdentificationSensor` basically provides the functionality of switching between learning faces online and recognizing faces. Due to the very limited resources of the robot platform in terms of CPU and memory the functionality of activating/deactivating was integrated. Team ToBI finished 8th place on their first appearance in the world cup.

2010: Middleware Abstraction & extended services After having established the *sensor/actuator* abstraction in 2009 and analysing the problems that have occurred during the competition it became clear that actually crashing components have caused a number of errors. Unsurprisingly not all the components are equally well tested and may be considered as stable on a research platform such as the one used by team ToBI during the competition. Almost all software components where under development and the team members are normally not the experts in any of the specific components used. The middleware used at the time (XCF, see Sec. 3.3) did enable a *loose coupling* of the components, which does increase the stability of the system since an error or crash of a single component does not affect other components as much. For the *robot behavior*, however, this means that data needed to go on with a certain part of the behavior may become unavailable. The

robot gets stuck in its current task.

The system components normally can be restarted automatically after a crash at runtime but the middleware did not support auto-reconnecting to the components. To fully detach the middleware from the behavior code inside Bonsai the complete Bonsai core was re-implemented to improve the middleware abstraction and add missing functionality necessary to improve the stability of the behavior code inside Bonsai, which included auto-reconnecting *sensors* and *actuators* if the connection was lost e.g. after a crash (see Sec. 5.1.1). This greatly improved the stability of the *robot behavior* and resulted in an 7^{th} place in the world cup.

2011: Strategies With the increasing complexity of tasks in the RoboCup@HOME league it became necessary to configure a certain Robot Behavior according to the current situation. For example: If the robot is supposed to autonomously explore its surroundings it may use a rather simple open space exploration (see [146]) purely based on the current laser readings of the system. For a person follow it is necessary to determine what kind of distance the robot should allow between itself and the person guiding. The computation of these factors for *robot behavior* is basically always the same but may vary in certain parameters. To enable a dynamic and context-sensitive adaption of *robot behavior* at runtime the Bonsai *strategies* where introduced (see Sec. 4.2.6). As a result of the experience in the @HOME league the concept of *strategies* was further extended to *informed strategies*, enabling more complex information fusion operations and adding more *sensor* input which allows to execute *robot behavior* according to semantic context information generated from the available *sensor* data. With these *strategies* at hand team ToBI was able to finish 5^{th} place in Istanbul 2011.

2012: Skills & SCXML In that year reusability of *robot behavior* became an important factor. Meanwhile a number of building blocks of *robot behavior* where available in Bonsai but to be actually reusable in other scenarios and tasks it was necessary to reduce scenario-specific code inside the building blocks. This means that details of the RoboCup rulebook where encoded in the building blocks of *robot behavior*, e.g. number of persons to search for in a person search behavior. To address this problem the Bonsai *skills* and the SCXML state machine abstraction (see Sec. 5.2) was introduced. The basic approach is to break up *robot behavior* in smaller, more reusable pieces of code (*skills*) and compose them using the SCXML description.

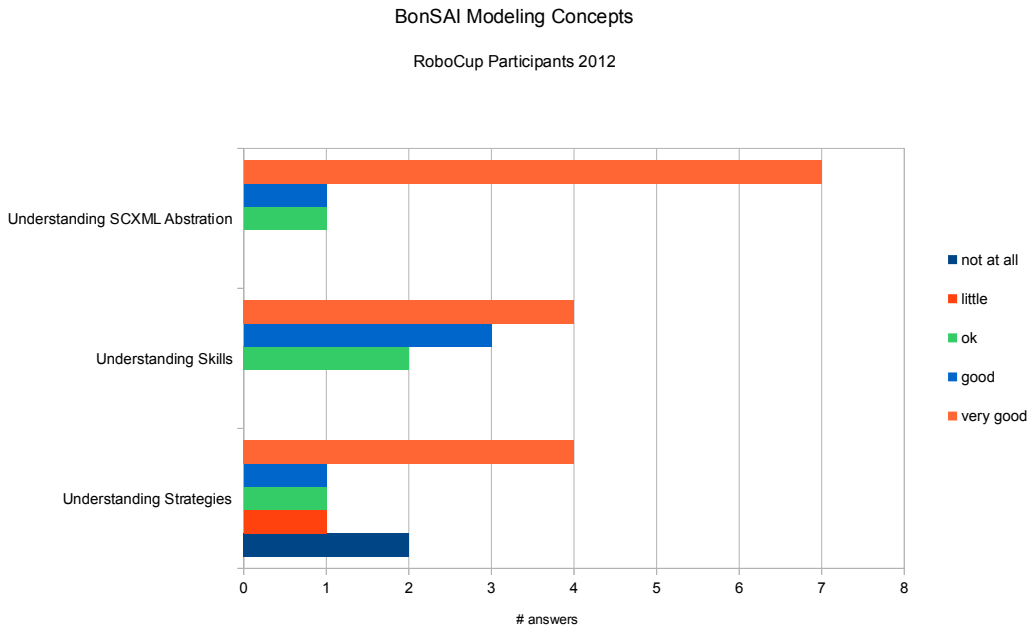


Figure 6.4.: Assessment of the RoboCup participants 2012 of the additional Bonsai modeling concept.

Results

Coming back to the two main questions covered by this rather long an uncommon type of evaluation, whether the behavior building blocks inside Bonsai can be improved over time by integrating experience gained from the RoboCup competition and if the Bonsai framework is usable by untrained developers, it is save to say that the students participating in the RoboCup as team ToBI had no problems with using Bonsai to compose *robot behavior*. The chosen nomenclature and the according building blocks, as visible in Table 6.2 and in Fig. 6.4, where well understood and used by the students. The 2009/10 data are shown consolidated because the data was acquired with the same online questionnaire in 2010.

In this regard it is important to mention that even though the overall system understanding is not as high (see Table 6.2), all participants feel very confident to be able to fix errors in the *robot behavior*. This means that the used robot still is a complex system with many components but the structure of Bonsai seems to make it easy to find and fix errors, especially for developers that have never been working with the system before.

The students where additionally asked to assess usability features of Bonsai (same features as in Sec. 6.3.2) based on their own experience. The combined

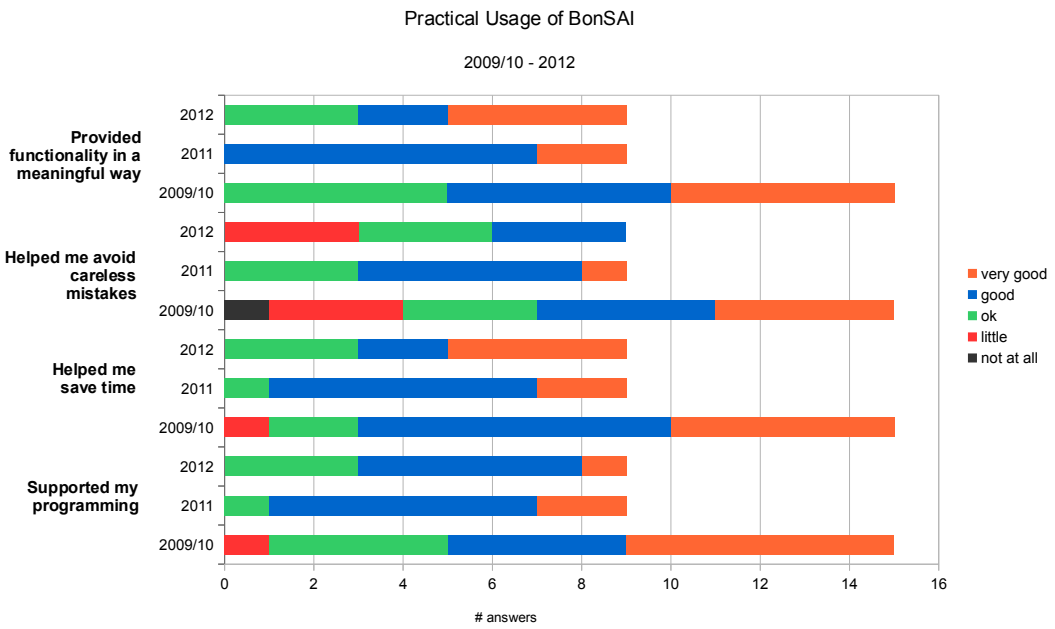


Figure 6.5.: Practical Usage Assessment of Bonsai during the RoboCup competition from 2009/10-2012. The according questions (**bold**) can be found on the right for each time period (2009/10, 2011 and 2012)

results from 2009/10-2012 can be found in Fig. 6.5. It shows that the two most important aspects of Bonsai in this context where reducing the time to compose a *robot behavior* and providing functionality of the robot in a meaningful way. The latter also corresponds to the high confidence in fixing errors. In terms of saving time it was possible to compose a complete new *robot behavior* (Finals 2012) and integrate a new *skill* (opening a door) on location in Mexico within only a few hours and successfully perform the task live in front of the audience. As said earlier, the competition itself is an evaluation of Bonsai and the system. As indicated in Table 6.2 team ToBI was able to improve every year while the competition got harder (more complex tasks, more competitors), the rules/tasks where changed twice (2010 and 2012) and the team members changed every year. This is an indicator that the capabilities of the system where well retained within Bonsai and could be improved over the years.

6.3. Portability & Usability: Bonsai Developer Studies

The Bonsai framework was developed to provide capsulated functionality of a robot platform, detached from underlying system components and middleware. An important factor was to identify commonalities of different scenarios and platforms to produce behavior code that can be reused more easily between those.

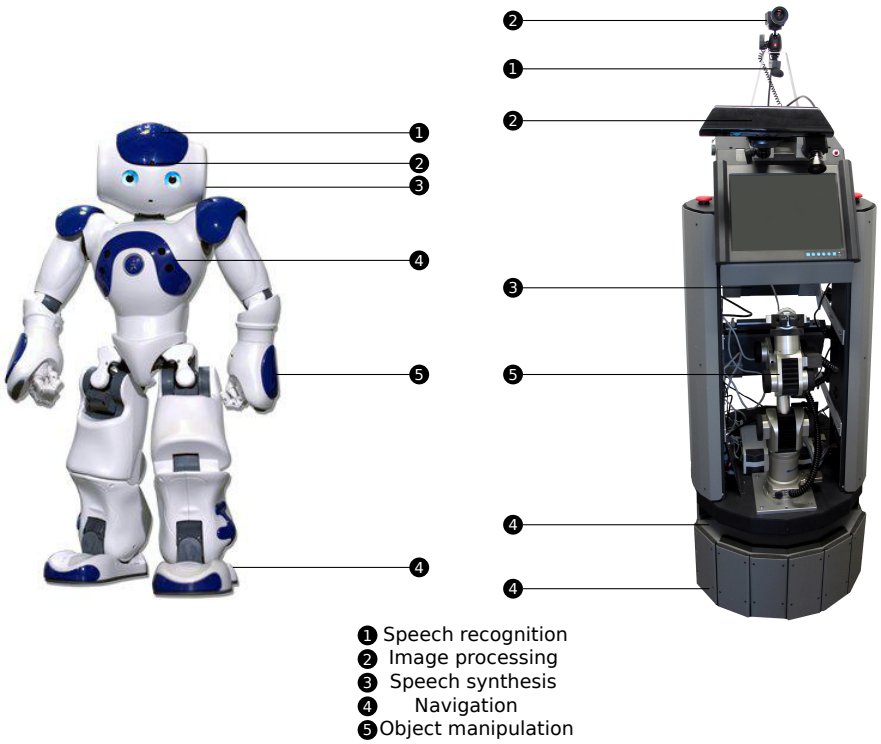


Figure 6.6.: The different functionality provided by the hardware of the NAO robot (left) and the BIRON platform (right): Speech recognition (1), Image processing (2), Speech synthesis (3), Navigation (4), Object manipulation (5).

To verify the suitability of the chosen level of abstraction for this purpose the Bonsai *skills* developed during the RoboCup competition where ported to a different platform, in this case the NAO robot [42], with the condition that the behavior code must be untouched. Hence, only the implementation of the *sensors* and *actuators* for the new platform where allowed to be adapted. The examination and integration was part of the bachelor thesis of Sebastian Schneider [114] that was successfully completed in 2010.

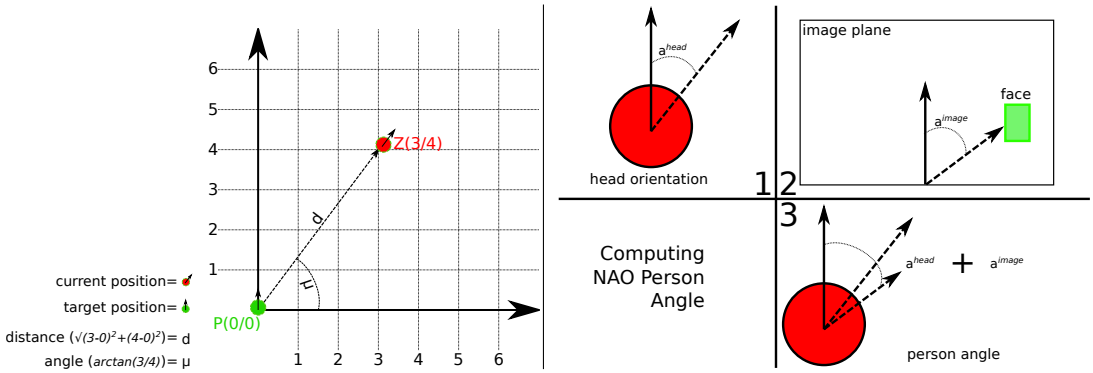


Figure 6.7.: **left:** Example for calculating the NAO odometry. **right:** Schema for computing the person angle with the NAO platform.

6.3.1. Bonsai Portability

The driving questions for this experiment where: "*Is it possible to integrate a new robotic platform into the existing architecture?*" and "*Is it necessary to change the existing architecture to achieve this?*".

Procedure

For comparing the System Behavior on the two different platforms the "Who is Who" task of the 2009 RoboCup [94] was chosen. In this task the robot has to find people inside an apartment and after detecting a person the robot has to learn their names and faces. One of the persons in the room is already known to the robot. After the searching phase the robot is going back to the entrance of the flat and all detected persons step in front of the robot. The robot then has to recognise their faces and say the correct name of the person. It is important to mention that this task is possible to perform without a SLAM running on the robot. There are versions of the NAO available that have an additional laser sensor for mapping and more recently there also is a version with a 3D sensor available. At the time when this test was conducted only the standard version was available. Hence, only tasks without SLAM could be taken into account.

The first step was to identify the Bonsai functionality within the new platform. In Fig. 6.6 the different functionality provided by the different platforms is shown. This is the first level of abstraction, both platforms can provide a similar functionality. The next step is to identify the *sensors* and *actuators* used within the Bonsai *skills* with the main challenge of providing all necessary information from the limited sensory data available on the NAO platform, namely the `Odometry` data and the `Person` data which allow the robot to track its position and detect

persons in its vicinity. Both information could be provided by adding a piece of software to the component level (see Fig. 4.1).

The NAO platform is a two-legged platform that does not provide odometry information. The according component can interpolate this information by counting the steps the robot has taken. On startup the navigation component will mark the current position as (0/0). The NAO robot can perform so called *walk patterns*, predefined sequences of movements of the legs to navigate the robot. The accuracy of the execution of this pattern defines the accuracy of the odometry. This is similar to the odometry information of the BIRON platform, where wheel rotation is counted to estimate traveled distances.

In case of the person data it was possible to follow a similar approach: The person information of the BIRON platform consisted of face information, including the distance and the angle of a person towards the robot and if the person is looking at the robot ("`gazing`) or not. Most of the information could be provided easily by the internal face tracking of the NAO platform.

The person distance and angle of the person with respect to the robot could be gained from the face tracking by taking the position and size of the faces in the camera image of the NAO into account. With an average face size in a fixed distance (e.g. 1 meter) it is easily possible to compute the distance of the person. For the person angle it was necessary to take the head orientation of the NAO robot into account. Since the optical center of the camera is calibrated to 0 degree of the head of the robot this computation comes down to add the head angle and the measured image angle of the detected face. A schematic overview can be found in on the right in Fig. 6.7. All other components either already used the same data structure as the BIRON platform because of high reuse of components between the labs or could be adapted in the same manner as described for the odometry/person data.

Results

After some first tests of the Robot Behavior on the NAO platform it became obvious that the scenario for the NAO had to be adapted due to an unexpected problem: The NAO is only 60cm tall, which means that it can only see the faces of people standing around the robot from a certain distance. This distance (approx. 1.5m) is problematic in an indoor environment because the person must neither be too far away nor to close for the integrated sonar sensor of the platform. This problem could have been overcome by changing the according *skills* but this was prohibited by the experimental settings.

To be able to still search for persons without changing the actual behavior we decided to conduct the tests with all persons in the room sitting down. In the successive rules of the RoboCup competition, robots had to be able to detect

persons that where sitting, which means this was only a minor change to the scenario with no effects on the actual behavior.

Apart from that the *sensor/actuator* abstraction of Bonsai made it possible to port the *robot behavior* from one platform to another without changing the behavior code. Underlaying components that may become necessary as explained earlier can be easily integrated into the system by following an information-driven integration (see Sec. 3.3) approach of Bonsai. Coming back to the original questions it has been shown that it is possible to integrate a new robotic platform into the existing architecture and the existing architecture supports the developer in doing so. Thus, it is not necessary to change the existing architecture because the chosen level of abstraction of the Bonsai *sensors* and *actuators* does provide sufficient unification and ability to support different robotic platforms.

6.3.2. Bonsai Usability

In contrast to the prior studies, which mainly focused on the improvement of the *robot behavior* itself, this part covers an additional aspect of the Bonsai framework: The usability for a developer of a *robot behavior*. The framework should be able to hide the complexity of underlying processes as described in Sec. 4.2.4 and 4.2.5. Hence, the driving question behind this study was: "*Can developers with no experience with Bonsai create robot behavior with little introduction in a short period of time?*". This is relevant for different reasons. The first is that for a framework that aims to enable continuous evaluation and improvement of *robot behavior* it is crucial that the chosen interfaces are easily comprehensible. To enable a developer to understand how to produce a certain behavior of the robot and the same time get an idea of where to check if something unexpected happens is maybe the most important factor in terms of usability for a developer of such a system.

Another reason why this is important for a framework like Bonsai is rapid prototyping. In robotics research, especially in the behavior context, rapid prototyping often is the only way to test if a certain behavior or part of a behavior works as expected. This is especially true for qualities that can't be simulated properly, e.g. user interaction scenarios.

Participants

The subjects where selected among students of computer science from the Bielefeld University that had good Java knowledge and where familiar with XML. There where 17 subjects with an average age of 27.8 years ranging from 24 to 32 years. The average knowledge of the BIRON system on a scale from 1 *not at all* to 5 *very well* was 2.3. The programming experience of the subjects can be found in Fig.

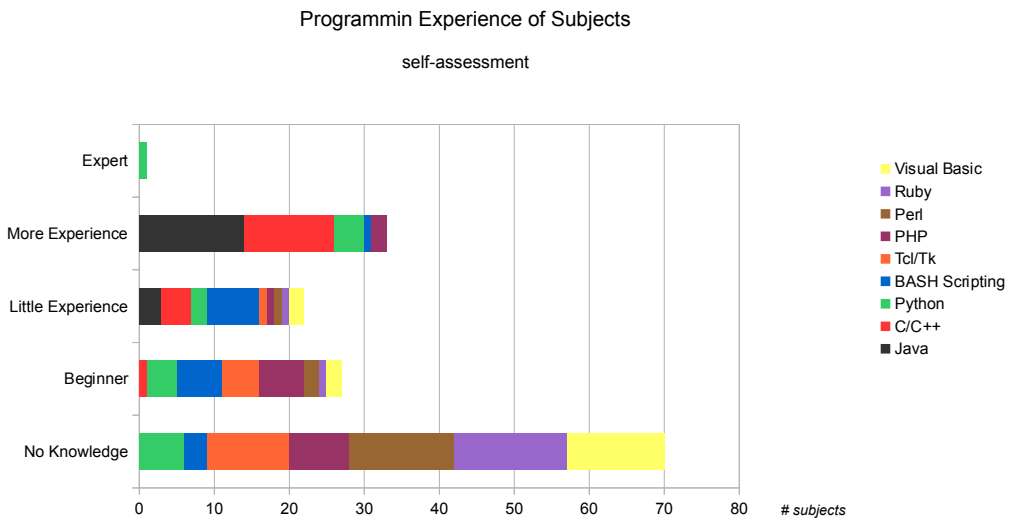


Figure 6.8.: The programming experience of the subjects of the Bonsai Study in 2012.

6.8, which shows that all except 3 subjects consider themselves as *more experienced* in Java, none had no experience. Interestingly no one considered himself as *expert*. All subjects had developed software in a team before, none of the participants had used Bonsai with SCXML before and only one had been working with the BIRON system but not with Bonsai.

Procedure

All subjects where informed that participation was voluntary and that they could stop at any point without giving a reason. After that they had to fill out a short questionnaire for personal data before they started the assignment. After filling out the first questionnaire the two pages of instructions where handed to the participants. They included a short overview of the Bonsai framework and its building blocks as well as the actual programming task. The original instruction sheets can be found in B.2.

The task of what the robot should do was described as follows: The robot should wait in front of the entrance door until it is open. After that the robot should drive to the kitchen and check for persons in the robots vicinity. If a person was detected the robot should announce that and leave the apartment. It was suggested that the participants start with creating an SCXML file for the overall task and after that program the necessary CheckForPerson *skill* for the task.

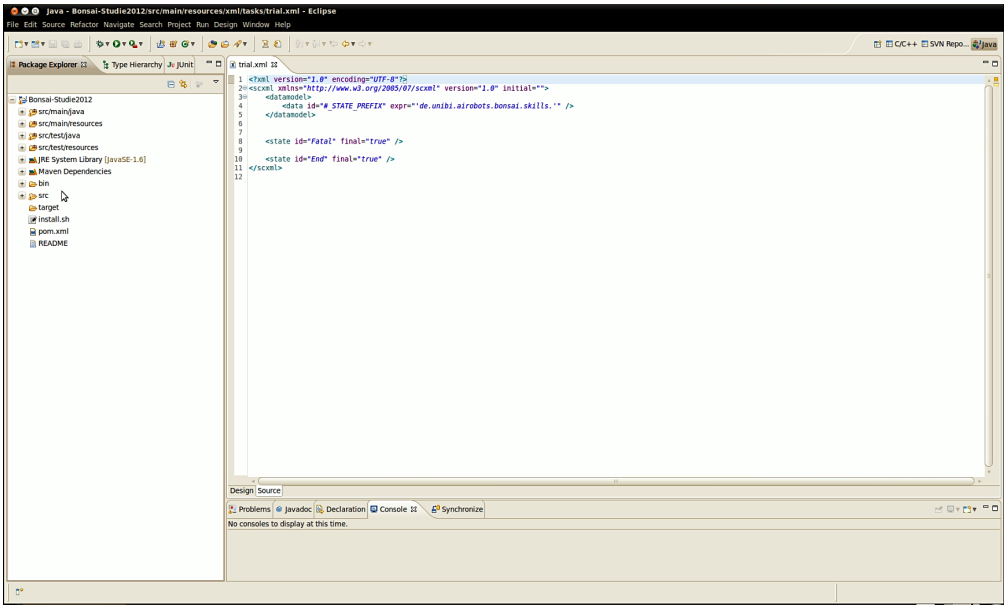


Figure 6.9.: The programming environment eclipse as it was prepared for the participants. Image taken from the original screen capture during the study.

For all parts, except the checking for persons, there where *skills* available in Bonsai but the participants had to look for them themselves. For the navigation there was an annotation file of the apartment prepared with example code on how to use the annotation within Bonsai. Only help given at that point was where to look for example code and how the annotation looked like and could be used. For that purpose there where two additional help sheets that could be used by all participants (laying next to the monitor) that included a list of all *sensors/actuators* available as well as a short intro on how to use SCXML, the annotation file and the *State Machine Viewer* GUI (see Fig. 6.10 on the left). All participants where given the same programming environment (see Fig. 6.9) with an empty SCXML file and an open Bonsai project. The simulation for testing was already running on the same computer, which means all participants could focus on the task and test by simply starting the State Machine Viewer via the *Run* option of the programming environment. The original help sheets can be found in B.3.

After successfully implementing the task and testing it in simulation the participants had to fill out a second questionnaire about their experience with Bonsai. The overall assignment took between 60 and 90min for each participant, including all questionnaires.

Kolmogorov-Smirnov-Test	Personal Data	Questionnaire
Standard Deviation	0.809	0.717
Median	4	1
Average Absolute Deviation from Median	0.647	0.529
P	0.23	0.01

Table 6.3.: Results of the Kolmogorov-Smirnov test for 'Complexity to create robot behavior' question before & after using Bonsai.

Results

The first and one of the most important results is that none of the participants could not finish the test. This is important because it means that it is possible without experience to code a complete *robot behavior* with Bonsai. For the second questionnaire the same scale from 1 to 5 was used as before in the RoboCup evaluation. The average understanding of the concepts behind Bonsai where very good: Average understanding of *sensors/actuators* was 4.41. The average understanding of *skills* was 4.35. The participants also where pretty confident about what they where doing, the average for the question "*How well do you think have you solved the assignment?*" was 4.24.

In terms of complexity Bonsai was also perceived very well. All participants where asked prior to the study how complex they think, from their own experience, the creation of an overall *robot behavior* is. Prior to the study it was an average of 4.18. On the second questionnaire they where asked how complex the creation of the overall *robot behavior* was with Bonsai, which resulted in an average of 1.53.

To check these numbers, due to the rather small number of participants, a *two-sample Kolmogorov-Smirnov test* was used. As the results in Table 6.3.2 indicate, the sample population cannot be assumed to be normally distributed. This means to test these numbers if they have occurred by chance alone (statistical hypothesis test) the Wilcoxon Signed Rank Test should be used. The results from a Wilcoxon Signed Rank Test are $W = 136, n_{s/r} = 16, z = 3.5$, which indicates that the numbers are statistically significant. In other words: The improvement of the complexity value before (4.18) and after (1.53) the assignment follows from the Bonsai experience and not from chance.

The second questionnaire also included an assessment of the programming functionality of the Bonsai framework by the participants similar to the one done with the RoboCup team ToBI. The participants where asked to rank Bonsai from their experience in the seven categories visible in Fig. 6.11. The scale was from 1 *not at all* to 5 *very much*.

Two of the questions stand out a little: The first one is "*Has Bonsai led to prob-*

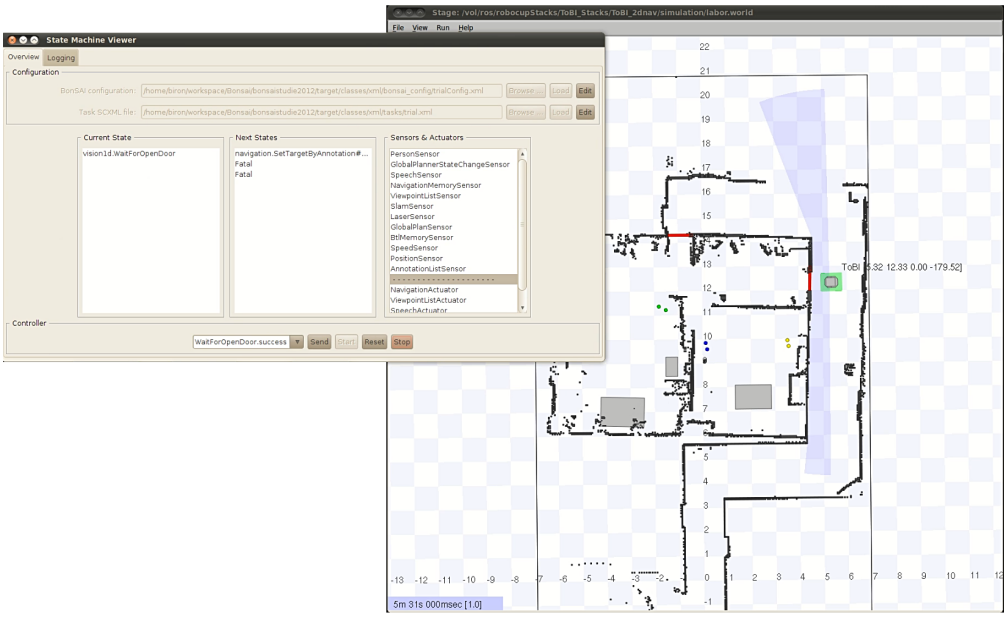


Figure 6.10.: The simulated test environment (background right) and the SCXML starter GUI (left front). Image taken from the original screen capture during the study.

lems because of errors?" and the second one is "*Has Bonsai provided functionality in a meaningful way?*". For both questions the anwers have been very positive, 12 people said that no problems at all occurred and the remaining 5 people still rated for rather no problems. The key factor here certainly was that the participants could focus on their programming with Bonsai, the necessary simulation was already running. But this also underlines the continuous improvement of the framework since its initiation in 2008. The 5 people answering rather not may have been the ones where a component of the simulation needed to be restarted before the last test but this was clearly not an error of the Bonsai framework. The second question is very positive, all participants agreed that Bonsai provides functionality in a meaningful way. This again underlines the well chosen level of abstraction as well as the chosen nomenclature within the Bonsai framework that, apart from enabling the structuring of the *robot behavior*, was meant to be easily comprehensible by developers.

In terms of re-usability the participants perceived Bonsai as helpful, 15 participants say Bonsai helped them to produce re-usable code and all but one said that Bonsai can help saving time creating a *robot behavior*. There was one participant who was actually uncertain because he had never developed a *robot behavior* before and could not say if approx. 60min was fast or not.

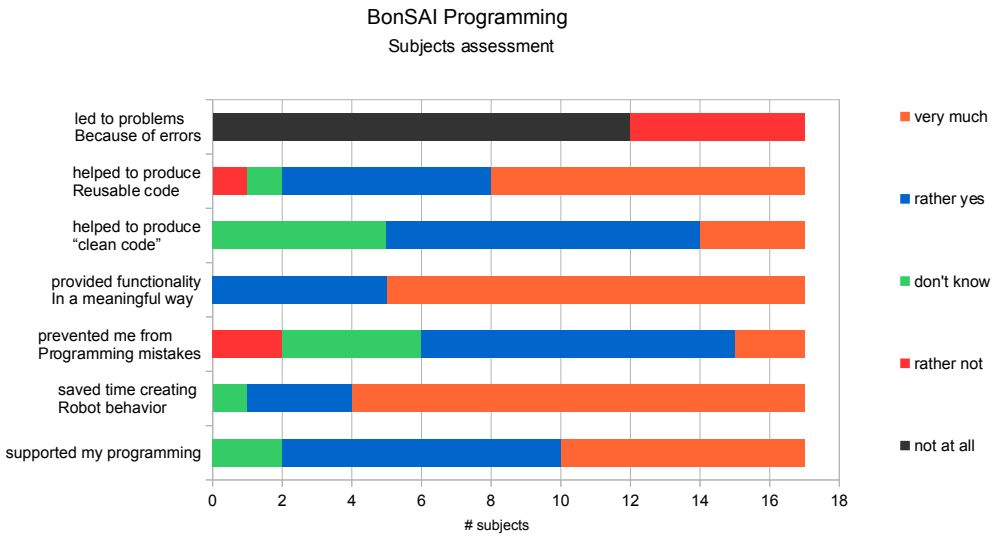


Figure 6.11.: The subjects' assessment of the programming functionality of the Bonsai framework.

In the second part of the questionnaire the participants where asked if they think, again from their own experience, if Bonsai was applicable on other robots or in other scenarios and if Bonsai was suitable for complex scenarios. Apart from 3 participants who where not sure all participants feel that Bonsai is suitable for other and more complex scenarios and can be used on other robots, which is congruent with my findings from other evaluations (see Sec. 6.3.1). Finally all participants where asked how well they think they have solved the programming task themselves on a scale from 1 *not at all* to 5 *very well*. Five where unsure (3), the other 12 participants thought they have done well (nine answered 4, three answered 5, average 4.24).

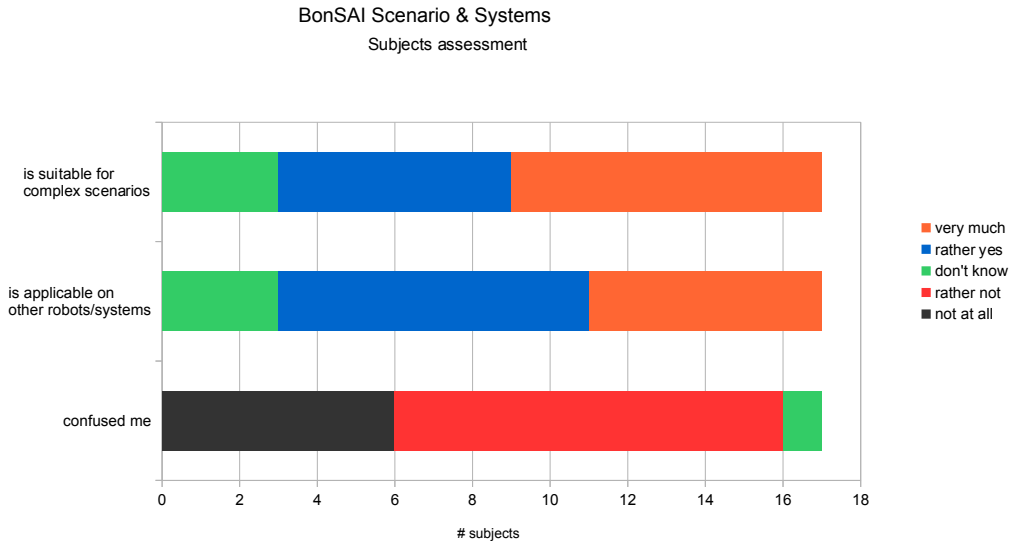


Figure 6.12.: The subjects' assessment of the applicability of the Bonsai framework.

7. Discussion & Conclusion

> In science, arguing about a definition and attempting to produce a single binary decision is often a waste of time.
>
> Aaron Sloman

In this thesis I have identified major the challenges when designing and improving an interactive system and the resulting challenges for developers and evaluators of such systems. Firstly the evaluators, coming from different areas of expertise, need to be able to easily compose a *robot behavior* and identify and locate errors of the *behavior modules* without requiring a deep knowledge about the whole system. Secondly the developers of such systems need to be able to implement improvements of the *robot behavior* from observations, possibly done by the evaluators, in real world interactions and easily extend the behavioral spectrum of the robot by rapidly prototyping new *behavior modules*.

For this matter it was necessary to abstract from the multitude of components within an interactive system and focus more on explicitly modeling the *robot behavior* instead of solely modeling the interplay of software components.

In the following section I will shortly discuss the architecture and the according implementation with regard to these perspectives and again look at some existing approaches followed by a more detailed discussion of the results from the different evaluations presented in Sec. 6. Finally I will give an outlook into some future challenges and possibilities with the Bonsai framework and mobile interactive systems at hand.

7.1. Interactive System Design with Bonsai

As I have pointed out in chapter 4 the development nowadays of interactive systems has changed its focus from putting together a system from scratch to evaluate and improve an existing system that comprises a number of different functionality and is deployable in different scenarios. The reason for this is obvious: There are more system components available from different areas of research with improved algorithms and they have gained a lot more functionalities over the past few years. Additionally there is new hardware available, for perception as well as

robot platforms. And lastly there is more computing power available to mobile platforms, allowing the execution of more components in parallel on a single platform, thereby enabling more complex scenarios and tasks that can be achieved with a single platform.

These developments have affected the architectural style for mobile interactive systems to a similar extent as *three layered architectures*, which still represents a fair amount of systems and frameworks (3T, CLARAty, OROCOS). As it was pointed out in Sec. 2.1.2, the *subsumption architecture* by Brooks [14], as one of the foundations for behavior-based robotics, was one of the first attempts to allow more complex systems to still be reactive but at the same time enable a task-based decomposition of actions. But as Hartley [49] pointed out it has pitfalls that make implementation of behavior-based robotic systems difficult with the mentioned changes but are still effecting the development of such systems today.

The main challenges for behavior-based robotic systems, based on the identified pitfalls of the *subsumption architecture*, are modularity, extensibility and execution context of the subsumption levels. Some of them have been clarified by Flynn [28] but with increasing complexity of robots and scenarios today they needed to be taken into consideration for the development of Bonsai. The existing frameworks today, as e.g. explained in Sec. 2.2, are doing a great job in connecting and controlling different software components of a system but barely support to build different robotic applications to e.g. perform in different scenarios. The resulting difficulty with modularity is that many software components are contributing to a single subsumption level (see Fig. 2.3) or even multiple levels but the interaction between the levels is fix. Brooks suggestion of reducing the communication between software components softens the problems but still makes a flexible combination of the levels of competence difficult. This is why *skills* in Bonsai are implementing *behavior modules* that models a functionality of the robot rather than a subsumption level.

The fixed interaction between the levels also accounts for the extensibility problem. Introducing new subsumption levels is only easily possible if they are added on top of the existing levels, implying a higher level of abstraction every time. In reality extensions in between the subsumption levels, e.g. for new capabilities or changes, result in changes in almost all levels of the system because they cannot be designed independently and become increasingly complex. Bonsai has introduced the *skill engine* that allows to model the control flow of the system (SCXML) independently of the *skills*, allowing to easily extend the behavioral spectrum of the robot.

The execution context within a classical behavior-based system refers to the problem of observing the environment and mapping it to a world state that triggers the activation of a behavior. But in fact the same observation can have different

implications depending on the context, which means different behaviors should be activated. Brooks suggested that behaviors should have to "second guess" before they are activated, e.g. by waiting on activation until necessary parts of the system have been idle for some time. With Bonsai this problem is tackled via different features: *Robot behavior* is composed via *skills*, which are not activated dependent on a specific state of the world. The control via SCXML explicitly allows to have a history for a *skill* (see Exc. 5.2.1) which represents an execution context of this specific *skill*. Additionally the Bonsai strategies allow for an in-depth analysis, or if you will multiple second guesses, of the information available to the whole system before actuation of the robot. And finally during the interrogation phase of a *skill* the necessary information is checked from a system memory, which is implicitly sensitive to e.g. changes from other components of the system. In combination this allows Bonsai to have a more flexible activation of *skills*.

From a developers point of view this flexibility often is referred to as the possibility of a framework to enable *rapid prototyping*. A definition can also be found in [54]:

> The ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.

For robot behavior this means that a developer needs to be able to put together a scenario from the existing system components in a short period of time. This enables the developer to test the code quickly and often implement new features more easily. As I introduced in Sec. 2.2.4, the *SMACH* library of ROS is currently very popular because it allows rapid prototyping of scenarios with the components provided via ROS. SMACH is a good example of how prototyping environments for robotic systems often come together: An existing middleware abstraction is extended by a state-machine-based tool that allows to sequence the different software components to produce a consistent system action. This often does not involve a modeling of the *robot behavior* itself, which is also true for SMACH, but rather implements a stateful sequencer of the system components. This is perfectly fine to test and improve features of the system components but is insufficient for modeling the *robot behavior* and test, improve or extend it over time. In fact, the typical way of rapid prototyping for robot scenarios as it is done with SMACH is counter productive for improving the *robot behavior* over time since it focuses on system features and does not involve a proper behavior abstraction. With SMACH the robot action and the control flow are modeled in the same place, the SMACH states. This reduces the reusability of the states and leaves the developers to implement robot behavior either inside the controller or in the system components themselves. The strong dependency on the ROS middleware and the ROS messages intensifies this effect. To improve *robot behavior* of interactive mobile robots

the user and the interaction with the user must be taken into account. This often implies to have a continuous evaluation cycle with real users which increases the importance of reusable *behavior modules* and independence of the middleware or scenario. This is why Bonsai emphasises the behavior abstraction and provides tools to model the *robot behavior* and the control flow separately. In the next section I will give some more details on the experiences from the different real world evaluations of the Bonsai framework.

7.2. Iterative Behavior Modeling: From User Studies to System Design

One of the main goals of this work was to improve behavior coordination for mobile interactive robots. Hence, an important factor, not only for this work but for mobile interactive robots in general, is the evaluation of the system in real world scenarios. As it was mentioned before, the simulation of user interaction is still difficult to do because simulation environments such as *morse* [1] or *V-Rep* [2] offer the possibility to have virtual humans inside the simulation but mostly to move them around the environment where they are rather used as "dynamic obstacles" than as real interaction partners for a robot.

But to improve the *robot behavior* in terms of interaction capabilities it was necessary for this thesis to make real world evaluation a part of the design process of Bonsai and enable developers to implement improvements gained from these interactions. This implies an iterative system design, as also pointed out by Lohse [74], with different scenarios as it was explained in Sec. 6.1 and 6.2. Some of the difficulties from the perspective of a system designer for such scenarios, e.g. with extensibility as described above, could only be uncovered by deploying the framework in the RoboCup@HOME scenario (see Exc. 6.1.1), which is based on the paradigm of regularly changing the scenario for the robot. The experience gained during the competition influenced the design of the Bonsai framework and has led to a 3rd place in the world championship in 2012 for the Team of Bielefeld.

This also stresses another important aspect during the development of the Bonsai framework that is crucial for systems that are continuously evaluated in real world scenarios: Reusability. For obvious reasons it is comfortable for developers when they do not have to re-code the software when they want to create a different *robot behavior*. In fact the usage of Bonsai saved a lot of time for the team members of ToBI and reduced the time for testing cycles. But this is not the only reason why reusability has become a factor for interactive mobile robots: When a

[1] `https://www.openrobots.org/wiki/morse`
[2] `http://coppeliarobotics.com/`

system is improved over time it is indispensable to keep the improved functionality somewhere in the system. If it is not reusable, the whole procedure of iterative design becomes inoperative. And lastly the challenges for dynamically changing the *robot behavior* and move away even further from pre-configured robot performances is even more difficult if the code that produces the *robot behavior* needs to be re-compiled for changes to take effect.

The resulting benefits from having this improved *behavior level* (see Fig. 4.1) available are also observable from the results of the *home tour* evaluation that was explained in Sec. 6.1. The quality measure that was introduced by Lohse [76], measuring the quality of an interaction based on so called *problem-related tasks* (see Tab. 6.1), helps to understand the benefit of the Bonsai *skills*: Problems during a real world interaction often result from unforeseen situations with a real user or difficulties from single components of the system that are not recoverable from the component level at that moment (e.g. object recognition). Once identified, Bonsai allows the implementation of strategies to cope with such situations and keep the interaction alive. Another common problem observed in the predecessor study (in 2007) during interactions was the transition between different interaction states of the system. With behavior code inside the system components it was necessary to switch control between many components, each of which had to keep track of the current interaction progress (see e.g. [122]). The recovery for situations like that, as e.g. described in [123], was to detect an error situation and then reset the control switch and restart the whole interaction. The user was able to reset the interaction by saying "*Reset*" at any time during an interaction. With Bonsai the behavior code was removed from the system components which makes the control switch between different components obsolete and allows to more flexibly compose the *robot behavior*. The improved recoverability with *skills* can also be observed from dramatic reduction of failure cases during the interaction in 6.1 from 2007 and 2010. This also allows us to improve the *System Grounding*, as e.g. introduced by [101], which means that the current state of the user interaction is aligned with the current state of the system. The "*reset* approach is problematic because it interrupts the interaction and makes flexible behavior composition difficult since the code that produces the robot actions is spread over many components and unwanted side effects when changing components is almost inevitable. Arguing from the results of the *Home Tour* and the RoboCup scenario, Bonsai did improve the interaction capabilities of the robot. The problem related tasks have been reduced and the flexible control made a "reset" of the system during the interaction unnecessary.

Another positive side effect of the Bonsai architecture was explained in Sec. 6.3.2: The usability of the framework. With the behavior code spread over the system it was difficult for novice developers to improve existing capabilities or

implement new features because they had to know a number of the system components in advance. The results of the usability evaluation indicates that it is possible to compose robust *robot behavior* even for users that have not been working with the system before. All participants where able to combine and program *skills* within approx. one hour, sometimes including the time to fill out the questionnaires. This is only possible via the separation of *skills* and the necessary control flow via the SCXML. In fact, if a new scenario can be handled with existing *skills* of the framework the task of the developer comes down to editing an XML file. This possibility has proven very helpful for the RoboCup team as well who sometimes need to put together a complete new scenario (e.g. when reaching the finals of the competition) within a couple of hours.

In spite of the improvements made with the Bonsai framework there are also certain limitations. As it was explained in Sec. 6.3.1 the Bonsai *skills* are generally portable to other platforms, however, the usage of the *skills* is limited by the hardware of the platform. This emphasises that, even though the *skills* are decoupled from e.g. middleware dependencies of a system, the system configuration still is an important part for the *skills* to work. Additionally Bonsai assumes, as mentioned in chapter 4, certain design paradigms to be fulfilled by the component layer, e.g. decoupled components and information-driven integration, which are useful for complex robotic systems and do improve the maintainability of the system. But this means that systems that do not fulfill these requirements can not make use of Bonsai *skills*. Also, even though the presented approach would also work on low level systems, the current implementation is not adequate for systems that have real-time requirements. All systems presented in this work where soft real-time for HRI, which means processing cycles of more than 100ms and less than 1sec. And lastly the Bonsai framework is capable of producing dynamic *robot behavior* to a certain extend but with the current implementation the developer ultimately designs the *skills* of the robot and the improvements. However, there are preliminaries to dynamically activate *skills* (see also next Sec. 7.3). This also implies that direct learning from an ongoing interaction is currently not done with Bonsai. But the level of abstraction and the existing *skills* are a practical basis.

7.3. Future Work

Challenges arising from the work with a mobile interactive robot have been described and tackled in this thesis. It is safe to say that the Bonsai framework in its current status provides a good basis for further improving the *robot behavior* and the interaction capabilities of systems, such as the BIRON system as well as others.

But the question remains: What else is now possible with Bonsai and may be

achieved in the future? For the short term, there are still technical improvements to further ease the work of the developers of *robot behavior*. The Bonsai skills already contain a representation of the system configuration, e.g. the available system components, that is evaluated by Bonsai on system startup. The configuration is defined by a developer in an XML config file (see A.2), which means it is done by hand, and is obviously a critical point (and a possible point of failure) for any system and behavior. A system check via Bonsai or the skill engine is possible that can automatically detect which components are running (service discovery) and instantiate the according skills.

Additionally the current implementation of Bonsai also includes lists of required *sensors* and *actuators* of each *skill* to check e.g. what can be run in parallel. To enable more dynamic activation of skills it is possible to have system components send requests for the activation of skills to the skill engine, similar to what is currently done with the dialog used on the BIRON/ToBI system (see [102]) based on the *task-state* pattern by Lütkebohle [78]. Based on methods described by Golombek [41] it would also be possible to introduce system health sensors and actuators to improve the robot behavior in situations where system components produce problems at runtime. This could be used to model the health status of the system and e.g. avoid usage of components that currently "hurt".

On a larger scale this could be the basis for investigating the impact of different arbitration mechanisms on more complex systems. As it was explained in Sec. 3.1, the system used during this thesis can be considered as discrete event system (see Sec. 2.1.1) with a state-based arbitration. The main question here is not how to do the arbitration, available methods range from command-fusion and priority-based approaches over Bayesian decision analysis to reinforcement learning and cover a research history of more than 30 years, but how these approaches influence the robot behavior. Bonsai could be used to test different approaches on the same system with the same behavior spectrum to minimize other factors than the arbitration mechanism.

And finally the ability to directly learn from interaction still is one of the biggest challenges for mobile interactive systems today. Currently we can distinguish two general scenarios for learning on the behavior level: The first scenario is to learn an action from the user that the robot then can do, e.g. handling of a tool or home furnishings. Existing approaches learn e.g. the movement of a toy [91] or how to use a pair of scissors [58]. The second scenario is to learn from observing the user to improve the interaction or help the user, e.g. detecting what the user does and providing help. If a robot could detect that the user is preparing a meal it could autonomously start to lay the table. Existing approaches use e.g. inverse reinforcement learning as a tool to recognize agents' behavior [108] but many approaches in scenarios like this use common-sense knowledge about the user's

actions. A mobile platform could help to further analyze a certain task and the recognition problem and identify relevant information, or processable information for a robot, that can be used to train a behavior recognition framework suitable for interactive robots.

A. Bonsai Implementation

Additional information about the implementation of the Bonsai framework. Complete Bonsai configuration files as well as Statechart XMLs that where used e.g. in the RoboCup competition are presented here.

A.1. Bonsai Configuration XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="options">
    <xsd:sequence>
      <xsd:element name="Option" minOccurs="0" maxOccurs="
          unbounded">
        <xsd:complexType>
          <xsd:simpleContent>
            <xsd:extension base="xsd:string">
              <xsd:attribute name="key" type="xsd:string" />
            </xsd:extension>
          </xsd:simpleContent>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="factoryoptions">
    <xsd:complexContent>
      <xsd:extension base="options">
        <xsd:attribute name="factoryClass" type="xsd:string" />
      </xsd:extension>
    </xsd:complexContent>
  </xsd:complexType>

  <!-- TODO this also needs an interface for complete middleware
      decoupling
    that can default to the standard sensor interface -->
  <xsd:complexType name="sensor">
    <xsd:sequence>
      <xsd:element name="Options" type="options" minOccurs="1"
```

```
31          maxOccurs="1" />
32      </xsd:sequence>
33      <xsd:attribute name="key" type="xsd:string" use="required" />
34      <xsd:attribute name="dataTypeClass" type="xsd:string"
35        use="required" />
36      <xsd:attribute name="listTypeClass" type="xsd:string"
37        use="optional" />
38      <xsd:attribute name="factoryClass" type="xsd:string"
39        use="required" />
40      <xsd:attribute name="sensorClass" type="xsd:string" use="
          required" />
41    </xsd:complexType>
42
43    <xsd:complexType name="actuator">
44      <xsd:sequence>
45        <xsd:element name="Options" type="options" minOccurs="1"
46          maxOccurs="1" />
47      </xsd:sequence>
48      <xsd:attribute name="key" type="xsd:string" use="required" />
49      <xsd:attribute name="factoryClass" type="xsd:string"
50        use="required" />
51      <xsd:attribute name="actuatorClass" type="xsd:string"
52        use="required" />
53      <xsd:attribute name="actuatorInterface" type="xsd:string"
54        use="required" />
55    </xsd:complexType>
56
57    <xsd:element name="BonsaiConfiguration">
58
59      <xsd:complexType>
60        <xsd:sequence>
61          <xsd:element name="FactoryOptions" type="factoryoptions"
62            minOccurs="0" maxOccurs="unbounded" />
63          <xsd:element name="Sensor" type="sensor" minOccurs="0"
64            maxOccurs="unbounded" />
65          <xsd:element name="Actuator" type="actuator" minOccurs="0"
66            maxOccurs="unbounded" />
67        </xsd:sequence>
68      </xsd:complexType>
69
70      <xsd:unique name="uniqueFactoryOptions">
71        <xsd:selector xpath="FactoryOptions" />
72        <xsd:field xpath="@className" />
73      </xsd:unique>
74
75      <xsd:unique name="uniqueSensors">
76        <xsd:selector xpath="Sensor" />
77        <xsd:field xpath="@key" />
78      </xsd:unique>
```

```
79
80      <xsd:unique name="uniqueActuators">
81        <xsd:selector xpath="Actuator" />
82        <xsd:field xpath="@key" />
83      </xsd:unique>
84
85    </xsd:element>
86
87  </xsd:schema>
```

Listing A.1: The XML schema for Bonsai configuration files.

A.2. Bonsai Configuration File Example

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <BonsaiConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-
       instance"
4    xsi:noNamespaceSchemaLocation="/vol/robocup/trunk/etc/schemas/
       BonsaiConfiguration.xsd">
5
6    <FactoryOptions factoryClass="de.unibi.citec.clf.bonsai.xcf.
       XcfFactory">
7      <Option key="errorOnInitialSubscription">true</Option>
8      <Option key="subscriberCheckInterval">30000</Option>
9      <Option key="remoteServerCheckInterval">30000</Option>
10   </FactoryOptions>
11
12   <!-- <Sensor key="PlaceSensor" dataTypeClass="de.unibi.airobots.
       btl.data.vision3d.SRPlaceData"
13     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
         sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
         XcfBtlPublisherSensor">
14     <Options> <Option key="publisherName">PlaceLabel</Option> </
         Options> </Sensor> -->
15   <Sensor key="ObjectSensor3D" dataTypeClass="de.unibi.airobots.
       btl.data.object.ObjectShapeList"
16     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
17     sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
         XcfBtlPublisherSensor">
18     <Options>
19       <Option key="publisherName">odcObject3D</Option>
20     </Options>
21   </Sensor>
22
23   <Sensor key="SlamSensor" dataTypeClass="de.unibi.airobots.btl.
       data.map.BinarySlamMap"
24     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
```

```
25      sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBinarySlamSensor">
26      <Options>
27        <Option key="publisherName">SlamMap</Option>
28      </Options>
29    </Sensor>
30    <Sensor key="objectsRecognized3DMemorySensor" dataTypeClass="de.
        unibi.airobots.btl.data.object.ObjectShapeList"
31      factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
            sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlMemorySensor">
32      <Options>
33        <Option key="memoryName">Scene</Option>
34        <Option key="xPath">objectsRecognized3D</Option>
35      </Options>
36    </Sensor>
37
38
39    <Sensor key="PlaneSensor" dataTypeClass="de.unibi.airobots.btl.
        data.vision3d.PlaneList"
40      factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
            sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlMemorySensor">
41      <Options>
42        <Option key="memoryName">ShortTerm</Option>
43      </Options>
44    </Sensor>
45    <Sensor key="LastPositionDataSensor" dataTypeClass="de.unibi.
        airobots.btl.data.navigation.PositionData"
46      factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
            sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlMemorySensor">
47      <Options>
48        <Option key="memoryName">Scene</Option>
49        <Option key="xPath">lastPositionData</Option>
50      </Options>
51    </Sensor>
52    <Sensor key="GlobalPlanSensor" dataTypeClass="de.unibi.airobots.
        btl.data.navigation.GlobalPlan"
53      factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
54      sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlPublisherSensor">
55      <Options>
56        <Option key="publisherName">GlobalPlan</Option>
57      </Options>
58    </Sensor>
59
60    <Sensor key="NavigationMemorySensor"
```

```
61      dataTypeClass="de.unibi.airobots.btl.data.navigation.
            NavigationGoalData"
62      factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
            sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlMemorySensor">
63      <Options>
64        <Option key="memoryName">Scene</Option>
65      </Options>
66    </Sensor>
67
68    <Sensor key="HandPosSensor" dataTypeClass="de.unibi.airobots.btl
          .data.person.HandPos"
69      factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
70      sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlPublisherSensor">
71      <Options>
72        <Option key="publisherName">HandPos</Option>
73      </Options>
74    </Sensor>
75
76    <Sensor key="GlobalPlannerStateChangeSensor"
77      dataTypeClass="de.unibi.airobots.btl.data.navigation.
            GlobalPlannerStateChange"
78      factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
79      sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlPublisherSensor">
80      <Options>
81        <Option key="publisherName">GlobalPlannerState</Option>
82      </Options>
83    </Sensor>
84    <Sensor key="PositionSensor" dataTypeClass="de.unibi.airobots.
          btl.data.navigation.PositionData"
85      factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
86      sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlPublisherSensor">
87      <Options>
88        <Option key="publisherName">interpolatedSlamPos</Option>
89      </Options>
90    </Sensor>
91
92    <Sensor key="PersonSensor" dataTypeClass="de.unibi.citec.clf.
          bonsai.sensors.data.PersonList"
93      factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
94      sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfActiveMemoryPersonSensor">
95      <Options>
96        <Option key="memoryName">Scene</Option>
97      </Options>
98    </Sensor>
```

```
99
100
101   <Sensor key="SpeedSensor" dataTypeClass="de.unibi.airobots.btl.
          data.navigation.SpeedData"
102     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
103     sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlPublisherSensor">
104     <Options>
105       <Option key="publisherName">SpeedData</Option>
106     </Options>
107   </Sensor>
108
109   <Sensor key="SpeechSensor" dataTypeClass="de.unibi.airobots.btl.
          data.speechrec.Utterance"
110     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
111     sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlPublisherSensor">
112     <Options>
113       <Option key="publisherName">isr</Option>
114     </Options>
115   </Sensor>
116   <Sensor key="ObjectSensor" dataTypeClass="de.unibi.airobots.btl.
          data.object.ObjectShapeList"
117     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
118     sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlPublisherSensor">
119     <Options>
120       <Option key="publisherName">odcObject</Option>
121     </Options>
122   </Sensor>
123
124   <Sensor key="LaserSensor" dataTypeClass="de.unibi.airobots.btl.
          data.vision1d.LaserData"
125     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
126     sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlPublisherSensor">
127     <Options>
128       <Option key="publisherName">LaserData</Option>
129     </Options>
130   </Sensor>
131
132   <Sensor key="SceneGridMapSensor" dataTypeClass="de.unibi.
          airobots.btl.data.map.DynamicGridMap"
133     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
134     sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfSceneGridMapSensor">
135     <Options>
136       <Option key="memoryName">Scene</Option>
137       <Option key="mapName">Planes</Option>
```

```
138     </Options>
139   </Sensor>
140
141   <Sensor key="AnnotationSensor" dataTypeClass="de.unibi.airobots.
         btl.data.map.Annotation"
142     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
           sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
           XcfBtlMemorySensor">
143     <Options>
144       <Option key="memoryName">Scene</Option>
145     </Options>
146   </Sensor>
147
148   <Sensor key="AnnotationListSensor" dataTypeClass="de.unibi.
         airobots.btl.data.map.Annotation"
149     listTypeClass="de.unibi.airobots.btl.List" factoryClass="de.
           unibi.citec.clf.bonsai.xcf.XcfFactory"
150     sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
           XcfBtlMemorySensor">
151     <Options>
152       <Option key="memoryName">Scene</Option>
153     </Options>
154   </Sensor>
155
156   <Sensor key="objectsRecognizedMemorySensor"
157     dataTypeClass="de.unibi.airobots.btl.data.object.
           ObjectShapeList"
158     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
           sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
           XcfBtlMemorySensor">
159     <Options>
160       <Option key="memoryName">Scene</Option>
161       <Option key="xPath">objectsRecognized</Option>
162     </Options>
163   </Sensor>
164
165   <Sensor key="graspedObjMemorySensor" dataTypeClass="de.unibi.
         airobots.btl.data.object.ObjectShapeData"
166     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
           sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
           XcfBtlMemorySensor">
167     <Options>
168       <Option key="memoryName">Scene</Option>
169       <Option key="xPath">graspedObj</Option>
170     </Options>
171   </Sensor>
172
173   <Sensor key="objectsRecognizedFilteredMemorySensor"
```

```
174     dataTypeClass="de.unibi.airobots.btl.data.object.
            ObjectShapeList"
175     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
            sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlMemorySensor">
176     <Options>
177       <Option key="memoryName">Scene</Option>
178       <Option key="xPath">objectsRecognizedFiltered</Option>
179     </Options>
180   </Sensor>
181
182   <Sensor key="betterPositionMemSensor" dataTypeClass="de.unibi.
        airobots.btl.data.navigation.PositionData"
183     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
184     sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlMemorySensor">
185     <Options>
186       <Option key="memoryName">Scene</Option>
187       <Option key="xPath">betterPositionToGrasp</Option>
188     </Options>
189   </Sensor>
190
191   <Sensor key="ClassfilterForGraspingSensor" dataTypeClass="de.
        unibi.airobots.btl.List"
192     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
            sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlMemorySensor">
193     <Options>
194       <Option key="memoryName">Scene</Option>
195       <Option key="xPath">classfilterForGrasping</Option>
196     </Options>
197   </Sensor>
198
199     <Sensor key="RefereePersonDataSensor" dataTypeClass="de.unibi.
            airobots.btl.data.person.PersonData"
200     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
            sensorClass="de.unibi.citec.clf.bonsai.xcf.sensors.
            XcfBtlMemorySensor">
201     <Options>
202       <Option key="memoryName">Scene</Option>
203       <Option key="xPath">refereePersonData</Option>
204     </Options>
205   </Sensor>
206
207
208   <Actuator key="IcewingChainSwitchActuator" factoryClass="de.
        unibi.citec.clf.bonsai.xcf.XcfFactory"
209     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
            IcewingChainSwitchActuator"
```

```
210     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfIcewingChainSwitchActuator">
211     <Options>
212       <Option key="serverName">OdcChainSwitchServer</Option>
213     </Options>
214   </Actuator>
215
216     <!-- memory actuators -->
217   <Actuator key="BtlMemoryAnnotationActuator" factoryClass="de.
        unibi.citec.clf.bonsai.xcf.XcfFactory"
218     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          BtlMemoryActuator"
219     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfBtlMemoryActuator">
220     <Options>
221       <Option key="serverName">Scene</Option>
222     </Options>
223   </Actuator>
224
225 <Actuator key="betterPositionMemAcutator" factoryClass="de.unibi.
      citec.clf.bonsai.xcf.XcfFactory"
226     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          BtlMemoryActuator"
227     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfBtlMemoryActuator">
228     <Options>
229       <Option key="serverName">Scene</Option>
230       <Option key="xPath">betterPositionToGrasp</Option>
231     </Options>
232   </Actuator>
233
234   <Actuator key="ClassfilterForGraspingActuator"
235     factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
236     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          BtlMemoryActuator"
237     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfBtlMemoryActuator">
238     <Options>
239       <Option key="serverName">Scene</Option>
240       <Option key="xPath">classfilterForGrasping</Option>
241     </Options>
242   </Actuator>
243
244     <Actuator key="objectsRecognizedMemoryActuator" factoryClass="
          de.unibi.citec.clf.bonsai.xcf.XcfFactory"
245     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          BtlMemoryActuator"
246     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfBtlMemoryActuator">
```

```
247     <Options>
248       <Option key="serverName">Scene</Option>
249       <Option key="xPath">objectsRecognized</Option>
250     </Options>
251   </Actuator>
252
253     <Actuator key="RefereePersonDataActuator" factoryClass="de.
          unibi.citec.clf.bonsai.xcf.XcfFactory"
254     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          BtlMemoryActuator"
255     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfBtlMemoryActuator">
256     <Options>
257       <Option key="serverName">Scene</Option>
258       <Option key="xPath">refereePersonData</Option>
259     </Options>
260   </Actuator>
261
262   <Actuator key="objectsRecognized3DMemoryActuator" factoryClass="
        de.unibi.citec.clf.bonsai.xcf.XcfFactory"
263     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          BtlMemoryActuator"
264     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfBtlMemoryActuator">
265     <Options>
266       <Option key="serverName">Scene</Option>
267       <Option key="xPath">objectsRecognized3D</Option>
268     </Options>
269   </Actuator>
270
271
272   <Actuator key="graspedObjMemoryReturnActuator" factoryClass="de.
        unibi.citec.clf.bonsai.xcf.XcfFactory"
273     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          BtlMemoryActuator"
274     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfBtlMemoryActuator">
275     <Options>
276       <Option key="serverName">Scene</Option>
277       <Option key="xPath">objReturnType</Option>
278     </Options>
279   </Actuator>
280
281   <Actuator key="BtlMemoryActuator" factoryClass="de.unibi.citec.
        clf.bonsai.xcf.XcfFactory"
282     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          BtlMemoryActuator"
283     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfBtlMemoryActuator">
```

```
284     <Options>
285       <Option key="serverName">Scene</Option>
286     </Options>
287   </Actuator>
288
289     <Actuator key="graspedObjMemoryActuator" factoryClass="de.
          unibi.citec.clf.bonsai.xcf.XcfFactory"
290     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          BtlMemoryActuator"
291     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfBtlMemoryActuator">
292     <Options>
293       <Option key="serverName">Scene</Option>
294       <Option key="xPath">graspedObj</Option>
295     </Options>
296   </Actuator>
297
298
299     <Actuator key="objectsRecognizedFilteredMemoryActuator"
          factoryClass="de.unibi.citec.clf.bonsai.xcf.XcfFactory"
300     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          BtlMemoryActuator"
301     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfBtlMemoryActuator">
302     <Options>
303       <Option key="serverName">Scene</Option>
304       <Option key="xPath">objectsRecognizedFiltered</Option>
305     </Options>
306   </Actuator>
307
308   <Actuator key="IsrActuator" factoryClass="de.unibi.citec.clf.
        bonsai.xcf.XcfFactory"
309     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          IsrActuator"
310     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfIsrActuator">
311     <Options>
312       <Option key="serverName">isr</Option>
313     </Options>
314   </Actuator>
315
316   <Actuator key="NavigationActuator" factoryClass="de.unibi.citec.
        clf.bonsai.xcf.XcfFactory"
317     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          NavigationActuator"
318     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfNavigationActuator">
319     <Options>
320       <Option key="serverName">Sunflower</Option>
```

```
321     </Options>
322   </Actuator>
323
324   <Actuator key="SpeechActuator" factoryClass="de.unibi.citec.clf.
        bonsai.xcf.XcfFactory"
325     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          SpeechActuator"
326     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfSpeechActuator">
327     <Options>
328       <Option key="serverName">SaySrv</Option>
329     </Options>
330   </Actuator>
331   <Actuator key="DynamicMoveBaseActuator" factoryClass="de.unibi.
        citec.clf.bonsai.xcf.XcfFactory"
332     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          ROSDynamicReconfigurationActuator"
333     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfROSDynamicReconfigurationActuator">
334     <Options>
335       <Option key="serverName">Sunflower</Option>
336       <Option key="methodNameRead">drcGetMoveBase</Option>
337       <Option key="methodNameWrite">drcSetMoveBase</Option>
338       <Option key="dataTypeClass">de.unibi.airobots.btl.data.
            navigation.ROSMoveBaseConfiguration</Option>
339     </Options>
340   </Actuator>
341
342   <Actuator key="SeamTargetActuator" factoryClass="de.unibi.citec.
        clf.bonsai.xcf.XcfFactory"
343     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          BtlMemoryActuator"
344     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfBtlMemoryActuator">
345     <Options>
346       <Option key="serverName">Scene</Option>
347     </Options>
348   </Actuator>
349
350   <Actuator key="DynamicBaseLocalPlannerActuator" factoryClass="de
        .unibi.citec.clf.bonsai.xcf.XcfFactory"
351     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          ROSDynamicReconfigurationActuator"
352     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfROSDynamicReconfigurationActuator">
353     <Options>
354       <Option key="serverName">Sunflower</Option>
355       <Option key="methodNameRead">drcGetTrajectoryPlanner</Option
            >
```

```
356       <Option key="methodNameWrite">drcSetTrajectoryPlanner</
              Option>
357       <Option key="dataTypeClass">de.unibi.airobots.btl.data.
              navigation.ROSBaseLocalPlannerConfiguration</Option>
358     </Options>
359   </Actuator>
360     <Actuator key="PoseActuatorTobi" factoryClass="de.unibi.citec.
          clf.bonsai.xcf.XcfFactory"
361     actuatorInterface="de.unibi.citec.clf.bonsai.actuators.
          PoseActuatorTobi"
362     actuatorClass="de.unibi.citec.clf.bonsai.xcf.actuators.
          XcfPoseActuatorTobi">
363     <Options>
364       <Option key="serverName">armControlServer</Option>
365     </Options>
366   </Actuator>
367
368 </BonsaiConfiguration>
```

Listing A.2: The Bonsai configuration file for the RoboCup finals in 2012.

A.3. Bonsai SCXML Finals 2012

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <scxml xmlns="http://www.w3.org/2005/07/scxml" version="1.0"
3    initial="openDoor">
4    <datamodel>
5      <data id="#_STATE_PREFIX" expr="'de.unibi.citec.clf.bonsai.
           skills.'" />
6    </datamodel>
7
8    <state id="openDoor" src="behaviors/openDoor.xml">
9      <transition event="OpenDoor.success" target="enterArena" />
10   </state>
11
12   <state id="enterArena" src="behaviors/
         enterArenaDirectlyAfterOpening.xml">
13     <transition event="success"
14       target="arm.HomeGripperAfterDoorOpen#beforCharging" />
15   </state>
16
17   <state id="arm.HomeGripperAfterDoorOpen#beforCharging">
18     <transition event="HomeGripperAfterDoorOpen.success"
19       target="dialog.Talk#driveOnStation" />
20     <transition event="HomeGripperAfterDoorOpen.fatal" target="
           Fatal" />
21   </state>
22
23   <state id="dialog.Talk#driveOnStation">
```

```
24      <datamodel>
25        <data id="#_MESSAGE" expr="'Going to charge.'" />
26        <data id="#_NONBLOCKING" expr="true" />
27      </datamodel>
28      <transition event="Talk.success" target="
           driveOntoChargingStation" />
29      <transition event="Talk.error" target="
           driveOntoChargingStation" />
30      <transition event="Talk.fatal" target="
           driveOntoChargingStation" />
31    </state>
32
33    <state id="driveOntoChargingStation" src="behaviors/
         driveOntoChargingStation.xml">
34      <onentry>
35        <send event="'driveOntoChargingStation.timeout'" delay="'60s
             '" />
36      </onentry>
37      <transition event="success" target="dialog.Talk#Charging" />
38      <transition event="fatal" target="Fatal" />
39      <transition event="driveOntoChargingStation.timeout"
40        target="dialog.Talk#Charging" />
41    </state>
42
43
44    <state id="dialog.Talk#Charging">
45      <datamodel>
46        <data id="#_MESSAGE" expr="'I am charging right now.'" />
47        <data id="#_NONBLOCKING" expr="true" />
48      </datamodel>
49      <transition event="Talk.success" target="final_ordering" />
50      <transition event="Talk.error" target="final_ordering" />
51      <transition event="Talk.fatal" target="final_ordering" />
52    </state>
53
54    <state id="final_ordering" src="behaviors/final_ordering.xml">
55      <transition event="success" target="tasks.finale.
           WaitForCleanUp" />
56      <transition event="fatal" target="tasks.finale.WaitForCleanUp"
           />
57    </state>
58
59    <state id="tasks.finale.WaitForCleanUp">
60      <transition event="WaitForCleanUp.success"
61        target="dialog.SimpleConfirmYesOrNo#validateCleanUp" />
62      <transition event="WaitForCleanUp.error"
63        target="dialog.SimpleConfirmYesOrNo#validateCleanUp" />
64      <transition event="WaitForCleanUp.fatal"
65        target="dialog.SimpleConfirmYesOrNo#validateCleanUp" />
```

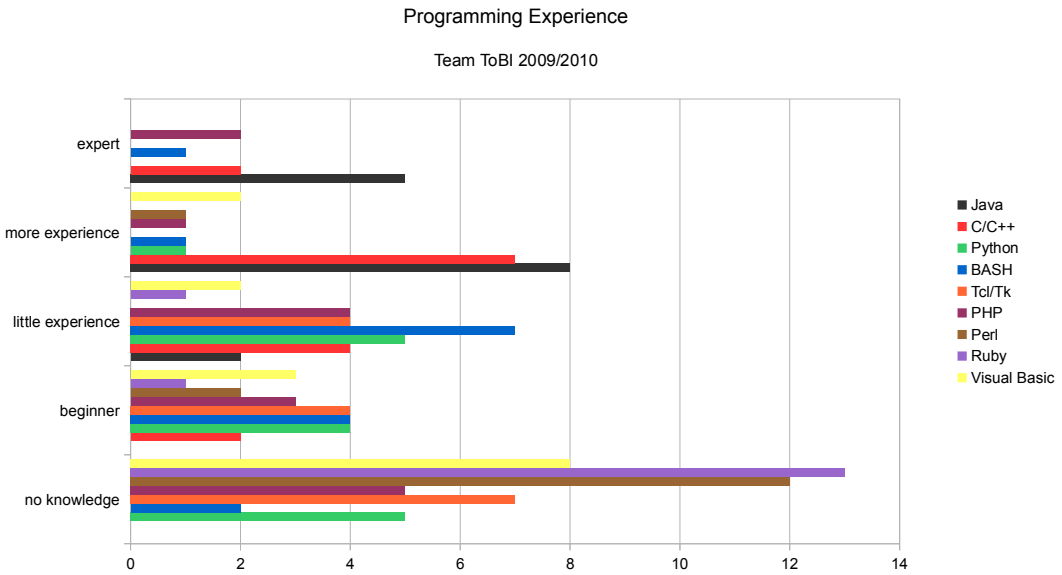
```
66    </state>
67
68    <state id="dialog.SimpleConfirmYesOrNo#validateCleanUp">
69      <transition event="SimpleConfirmYesOrNo.success.confirmYes"
70        target="mapWiping" />
71      <transition event="SimpleConfirmYesOrNo.success.confirmNo"
72        target="tasks.finale.WaitForCleanUp" />
73      <transition event="SimpleConfirmYesOrNo.fatal">
74        <send event="tasks.finale.WaitForCleanUp" />
75      </transition>
76    </state>
77
78    <state id="mapWiping" src="behaviors/sceneMapWiping.xml">
79      <transition event="success" target="dialog.Talk#done" />
80      <transition event="fatal" target="dialog.Talk#done" />
81    </state>
82
83    <state id="dialog.Talk#done">
84      <datamodel>
85        <data id="#_MESSAGE" expr="'Done.'" />
86        <data id="#_NONBLOCKING" expr="true" />
87      </datamodel>
88      <transition event="Talk.success" target="End" />
89      <transition event="Talk.error" target="End" />
90      <transition event="Talk.fatal" target="End" />
91    </state>
92
93    <!-- **********FAIL**END********** -->
94    <state id="Fatal" final="true" />
95
96    <state id="End" final="true" />
97  </scxml>
```

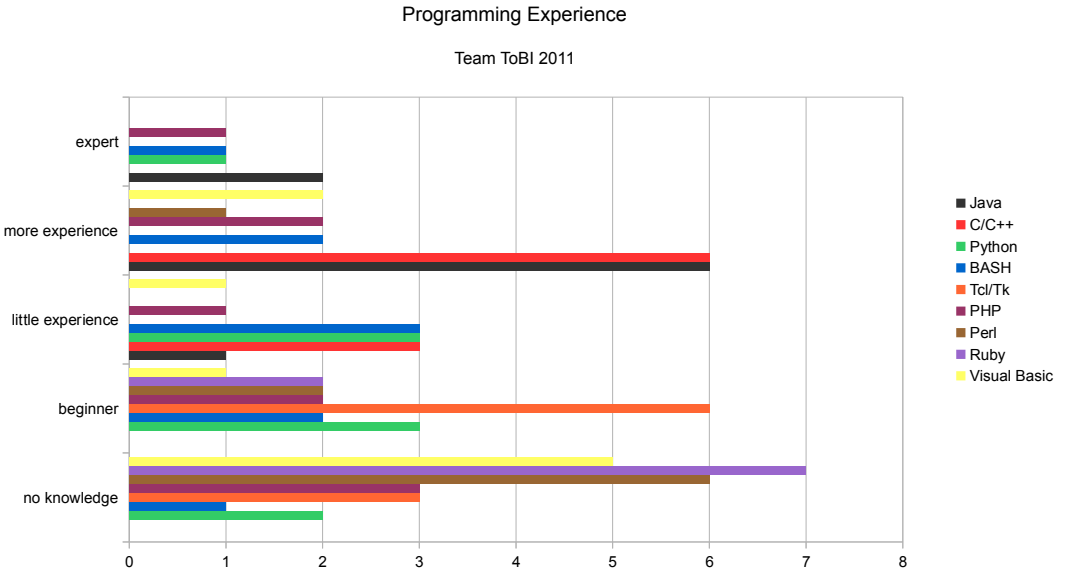
Listing A.3: The SCXML file of the task performed during the finals of the RoboCup 2012.

B. Bonsai Evaluation

Additional data/information about the Bonsai evaluation.

B.1. Team ToBI Programming Experience in 2009/2010





B.2. Usability Study Instruction Sheet

The instruction sheets that was given to the participants of the study.

BonSAI-Studie 2012: Erläuterungen

In der folgenden Studie wird es darum gehen ein Verhalten für den Roboter BIRON zu entwickeln. Dazu soll ein Ablauf aus bestehenden Fähigkeiten des Roboters (Skills) erstellt werden und ein neuer "Skill" entwickelt werden. Im Anschluss soll das neue Verhalten in der Simulation getestet werden. Um diese Aufgabe zu bewältigen werden Dir hier einige Grundkonzepte des BonSAI-Frameworks vorgestellt. Du kannst aber auch jederzeit mit dem Versuchsbegleiter sprechen und Fragen stellen. Versuche jedoch immer erst selbständig eine Lösung zu finden, Du kannst Dich dabei an den vorhandenen Skills und Beispielen orientieren. Im Anschluss möchten wir Dich bitten noch einen weiteren Fragebogen auszufüllen, in dem wir Deine Erfahrungen mit BonSAI erfassen wollen.

BonSAI:

Hierbei handelt es sich um eine Abstraktionsschicht für das Verhalten des Roboters, die in Java implementiert wurde. Es ist das Framework, mit dem Du gleich arbeiten wirst.

Komponente

Eine Komponente bezeichnet eine Software, die auf dem System läuft und entweder alleine oder in Kombination bestimmte Daten/Informationen/Aktionen zur Verfügung stellt. Du hast in der Regel nur mit den Daten der Komponenten zun tun → siehe BTL!

Bielefeld Type Library (BTL)

Die BTL ist eine Library, die Funktionalität zum Erstellen und Verarbeiten/Auslesen von XML-Daten, z.B. der Komponenten, zur Verfügung stellt. Die BTL übernimmt z.B. das parsing, behält dabei aber die Erweiterbarkeit von XML. Eine BTL-Datenstruktur beschreibt also ein Minimum an Information, das im System ausgetauscht wird. PersonData beschreibt z.B. die Informationen die mindestens enthalten sein müssen, damit das System mit einer Person interagieren kann.

SCXML:

Steht für State Chart XML und ist eine State Machine Notation die für die Abstraktion eines Programmablaufs (Control Flow) verwendet wird. Dabei werden die Zustände der Maschine/ Übergänge der Maschine als XML modelliert, um dann von einer Engine ausgeführt zu werden (z.B. innerhalb von BonSAI).

Sensor:

In BonSAI beschreibt ein Sensor eine Informationsquelle. Dabei kann es sich um einen echten Hardwaresensor, z.B. LaserSensor, handeln, oder aber um eine Kette von Komponenten, die eine bestimmte Aufgabe haben, z.B. PersonSensor. Sensoren liefern immer XML-Daten.

Actuator:

Innerhalb von BonSAI beschreibt ein Actuator ein ansprechbares/ausführbares Ziel, z.B. einen Hardware-Aktuator wie einen Greifarm (ArmActuator) oder aber eine Programmkette, die angestoßen werden muss um eine Aktion auszuführen, z.B. NavigationActuator. Ein Actuator bietet i.d.R. ein Set von Funktionen an, die z.B. mit einem XML-Dokument als Parameter aufgerufen werden können, z.B. navigationActuator.setGoal(NavigationGoalData data).

Skill:

Ein Skill ist die Beschreibung einer bestimmten Abfolge, die der Roboter ausführen kann, z.B. einer Person folgen. Ein Skill entspricht i.d.R. einem Zustand in der SCXML-Beschreibung und sollte möglichst nur eine Fähigkeit beschreiben. Loops, z.B. while(), sollten dringend vermieden werden.

Ein Skill besteht also aus einer Abfolge von Sensor/Actuator-Aufrufen um eine bestimmte Aufgabe zu erfüllen, z.B. einer Person folgen.

Aufgabenstellung:

Erstelle mit Hilfe von BonSAI folgenden Ablauf für den Roboter BIRON: “Der Roboter wartet vor der Eingangstür bis diese geöffnet wird und soll dann in die Küche fahren. In der Küche angekommen muss der Roboter nach Personen suchen und anzeigen, das er eine Person gefunden hat. Danach fährt der Roboter wieder zurück zum Eingang wo er die Aufgabe begonnen hat.” Alle benötigten Skills, bis auf die Personensuche, sind bereits in BonSAI enthalten.
1.) Erstelle bitte erst eine Statemachine mittels SCXML für den Gesamtablauf. Dabei kannst Du auf die bestehenden Skills zurückgreifen, eine Datei ist vorbereitet. Die Personensuche kannst Du erstmal raus lassen, da diese erst von Dir erstellt werden soll. Ein nachträgliches einfügen ist sehr einfach.
2.) Erstelle den “CheckPerson” Skill, in dem Du eine entsprechende Klasse in einem der bonsai.skill Pakete anlegst. In diesem Skill sollst Du überprüfen ob sich eine Person im Sichtfeld des Roboters befindet. Der Roboter soll dann durch sein Verhalten anzeigen, dass er eine Person gefunden hat, z.B. etwas sagen oder auf die Person zu fahren. Danach kannst Du die fertigen Skills verwenden um den Roboter in einen bestimmten Raum fahren zu lassen. Diesen Teil musst Du noch in Deinem SCXML ergänzen und testen, detailliertere Erläuterungen dazu findest Du auf dem Blatt ”Tips & Tricks”.

Du kannst Deinen Ablauf oder einzelne Skills zwischendurch immer in der Simulation testen.
Netbeans/Eclipse ist bereits fertig eingerichtet, genauso wie das BonSAI-Config-File. Du kannst Dich voll auf den neuen Skill und die SCXML konzentrieren.

Hilfe:

- Bereits existierende Skills findest Du im package *de.unibi.airobots.bonsai.skills*.
- Bereits existierende Strategies findest Du im package *de.unibi.airobots.bonsai.strategies.*
- Ein neuer Skill muss von der Klasse *AbstractState* erben.
- BTL-Daten haben die Funktion *fromElement(Element e, Data d)*, um ein Objekt aus einem XOM Element zu erstellen

B.3. Additional Help Sheet

BonSAI Tipps & Tricks

Hier werden Dir ein paar Tipps und Tricks bei der Programmierung mit BonSAI gegeben. Das ist keine Vorgabe wie Du programmieren sollst, aber es kann Dir ein bisschen Zeit (beim Suchen) ersparen. Du kannst aber auch erst selbstständig anfangen und Dir später die Tipps bei Bedarf angucken.

Eine Übersicht der verfügbaren Sensoren/Aktuatoren in BonSAI findest Du in der folgenden Tabelle:

Sensoren	Aktuatoren
LaserSensor	ViewpointListActuator
SpeechSensor	SpeechActuator
BtlMemorySensor	NavigationActuator
ViewpointListSensor	BtlMemoryActuator
PersonSensor	
PositionSensor	
AnnotationListSensor	
GlobalPlanSensor	
GlobalPlanSensor	
SpeedSensor	
SlamSensor	
GlobalPlannerStateChangeSensor	
NavigationMemorySensor	

SCXML

Der erste Teil der Aufgabe ist eine SCXML Datei zu erstellen. Eine XML-Datei ist bereits vorbereitet, die Du mit den States füllen musst. Ein State besteht immer aus einem <*state id=""*> TAG, die ID bezieht sich dabei auf den Skill den Du ausführen möchtest. Befindet sich der Skill *DriveToPosition* z.B. im Packet *navigation* wäre die id <*state id="navigation.DriveToPosition"*>. Innerhalb des States werden Transitions definiert, je nachdem was in einem Skill passiert: <*transition event="" target="" />* Es sind 3 Standardevents vorgegeben: *success, error, fatal*, die sich auf einen Skill beziehen müssen, also z.B. <*transition event="DriveToPosition.success" target="" />*, das Target gibt den State an, in den Du dann wechseln möchtest. Hier muss wieder eine ID rein, z.B. *dialog.Talk:*
<*transition event="DriveToPosition.success" target="dialog.Talk" />*. Das <state>-TAG kann wiederum States enthalten (substate) und auch noch das <*parallel id=""*>*-TAG*, das States enthalten kann die parallel ausgeführt werden. Die ID bezieht sich nicht auf die Skills, muss aber eindeutig sein, z.B. <*parallel id="FollowingPerson"*> <*state ….*>... </*parallel*>.

Um z.B. dem Talk-Skill den Text übergeben zu können, der gesagt werden soll, gibt es ein *<datamodel>*-Tag. In diesem lassen sich <data>-TAGs definieren, die durch eine ID zugeordnet werden. Folgedes Beispiel würde z.B. den Text “Hello World.” ausgeben:

<datamodel>
<data id=”#_MESSAGE” expr=”’Hello World.’” />
</datamodel>

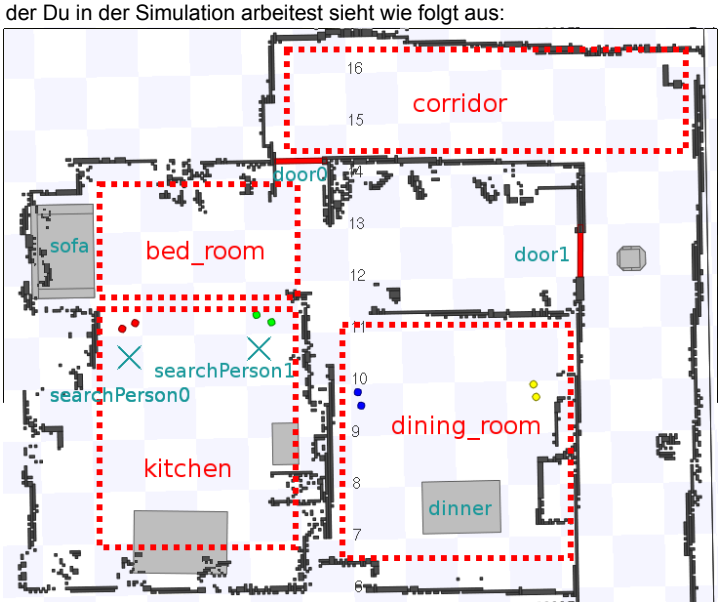
Um Daten in einem Skill auszulesen, z.B. aus einer Annotation einer Karte, wird ebenfalls das

<datamodel>-TAG verwendet:
<datamodel>
<data id=”#_ANNOTATION_LABEL” expr=”’kitchen’” />
</datamodel>

Der zugehörige Skill um an eine solche Position zu fahren heißt *navigation.SetTargetByAnnotation* und ist bereits vorhanden.

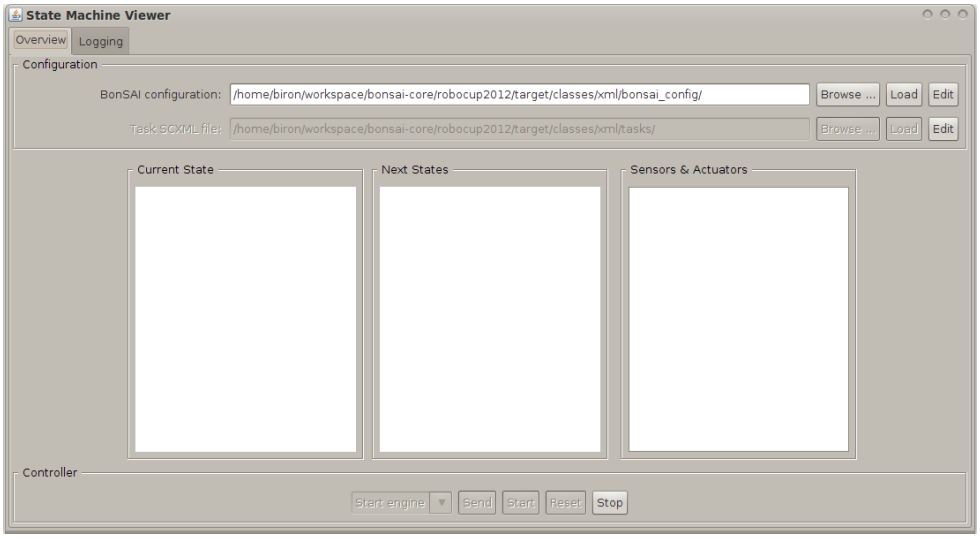
Annotation

Die Karte mit der Du in der Simulation arbeitest sieht wie folgt aus:



Die farblich markierten Bereiche entsprechen den vorhandenen Annotationen, ROT = *<data id=”#_ANNOTATION_LABEL” expr=”’kitchen’” />*, *BLAU = <data id=”#_VIEWPOIT_LABEL” expr=”searchPerson1” />*.

Testing



Die Simulation für das System läuft bereits im Hintergrund auf dem Laptop. Um den Ablauf testen zu können kannst Du in dem IDE Deiner Wahl (netbeans/eclipse) direkt auf das “Run Application” Symbol klicken, die Test-GUI ist voreingestellt. Wenn Du etwas mehr Kontrolle haben möchtest, kannst Du auch die Klasse unter *bonsai.engine.SCXMLStarter.java* als “Java-Application” ausführen lassen. In der GUI ist ein “Logging-Tab” zu sehen, in dem die Ausgaben aus Deinen Skills auftauchen. Im “Overview-Tab” findest Du die Einstellungen für die Ausführung, das BonSAI config ist bereits korrekt voreingestellt und muss mit “Load” geladen werden. Danach kann ein SCXML file geladen werden, ebenfalls durch “Load”. Tritt hier eine Fehlermeldung auf, im “Logging-Tab” sind dann Ddetails zu finden, ist das SCXML file nicht korrekt, z.B. eine ID nicht eindeutig oder der entsprechende Skill konnte nicht gefunden werden.
Wenn beide Dateien geladen sind kann mit den Schaltflächen unten (Send, Start, Reset, Stop) der Ablauf getestet werden. Die “Send” Schaltfläche erlaubt es einem Skill bestimmte Events zu schicken, was für diesen Versuch wahrscheinlich nicht notwendig ist. Bitte bedenke, das Du nach einem Stop den Roboter in der Simulation ggf. wieder in die Ausgangsposition bringen musst.

Bei einem Neustart der GUI musst Du ebenfalls den Roboter in der Simulation wieder in die Ausgangsposition schieben um den gesamten Ablauf testen zu können.
Dabei kann es sein, das die Lokalisation des Roboters auf der Karte verloren geht. Wir haben das Tool “rviz” mit gestartet, dort kannst Du, nachdem Du auf “2D Pose Estimate” geklickt hast, durch klicken auf die Karte in rviz die Position des Roboters vorgeben bzw. zurücksetzen. Bevor Du die Maustaste los lässt ziehst Du den Zeiger in die Richtung, in die der Roboter fährt, also
click=Position des Roboters, drag=Orientierung
des Roboters. Der Roboter kann nur dann auf eine globale Koordinate auf der Karte, z.B. aus einer Annotation, fahren, wenn er sich korrekt lokalisiert hat.

Bibliography

[1] K. Aoyama and H. Shimomura. Real World Speech Interaction with a Humanoid Robot on a Layered Robot Behavior Control Architecture. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, number April, pages 3814–3819, Barcelona, Spain, 2005.

[2] R. C. Arkin. Motor Schema-Based Mobile Robot Navigation. *The International journal of robotics research*, 8(4):92–112, 1989.

[3] R. C. Arkin. Towards the Unification of Navigational Planning and Reactive Control. Technical report, Menlo Park, Califoria, USA, 1989.

[4] R. C. Arkin. *Behavior-Based Robotics*. The MIT Press, Cambridge, Massachusetts, 1998.

[5] M. Asada and H. Kitano. The RoboCup Challenge. *Robotics and Autonomous Systems*, 29(1):3–12, Oct. 1999.

[6] T. Baier, M. Hüser, D. Westhoff, and J. Zhang. A flexible software architecture for multi-modal service robots. In *Computational Engineering in Systems Applications, IMACS Multiconference on*, volume 1, pages 587–592, 2006.

[7] J. Baltes. A benchmark suite for mobile robots. *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on.*, 2:1101–1106, 2000.

[8] J. Barnett, R. Akolkar, R. Auburn, M. Bodell, D. C. Burnett, J. Carter, S. McGlashan, T. Lager, M. Helbing, R. Hosn, T. Raman, K. Reifenrath, and N. Rosenthal. State Chart XML (SCXML): State Machine Notation for Control Abstraction, 2012.

[9] S. Behnke. Robot competitions-ideal benchmarks for robotics research. In *Proc. of IROS-2006 Workshop on Benchmarks in Robotics Research*, Beijing, China, 2006. IEEE/RSJ.

[10] M. Ben. *Principles of concurrent and distributed programming.* Addison-Wesley, 2nd edition, 2006.

[11] G. Biggs and B. MacDonald. A survey of robot programming systems. In *Proceedings of the Australasian conference on robotics and automation*, Brisbane, Australia, 2003.

[12] J. Bohren, R. B. Rusu, E. Gil Jones, E. Marder-Eppstein, C. Pantofaru, M. Wise, L. Mosenlechner, W. Meeussen, and S. Holzer. Towards autonomous robotic butlers: Lessons learned with the PR2. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on.*, pages 5568–5575, Shanghai, China, May 2011. IEEE.

[13] S. A. Brandt and G. J. Nutt. Flexible Soft Real-Time Processing in Middleware. *Real-Time Systems*, 22(1-2):77–118, 2002.

[14] R. Brooks. A robust layered control system for a mobile robot. *Robotics and Automation, IEEE Journal of*, 2(1):14–23, 1986.

[15] R. Brooks. Elephants don't play chess. *Robotics and autonomous systems*, 6(1):3–15, 1990.

[16] R. Brooks. Intelligence without Reason. In *Proceedings of IJCAI-91*, number 1293, pages 569–595, Sidney, Australia, 1991.

[17] D. Brugali and P. Scandurra. Component-based robotic engineering (part i). *Robotics \& Automation Magazine, IEEE*, 16(4):84—-96, 2009.

[18] D. Brugali and A. Shakhimardanov. Component-based Robotic Engineering (part II) Systems and Models. *Robotics & Automation Magazine, IEEE*, 17(1):100—-112, 2010.

[19] J. Bryson. The behavior-oriented design of modular agent intelligence. *Agent technologies, infrastructures, tools, and applications for e-Services*, pages 61–76, 2003.

[20] J. Bryson and L. A. Stein. Architectures and idioms: Making progress in agent design. *Intelligent Agents VII Agent Theories Architectures and Languages*, pages 73–88, 2001.

[21] J. Coakley and E. Dunning. *Handbook of sports studies.* Sage Publications Ltd, 2000.

[22] J. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proceedings of the 1992 IEEE Conference on Robotics and Automation (ICRA-92)*, pages 2719–2724, 1992.

[23] E. Coste-Manire and R. Simmons. Architecture , the Backbone of Robotic Systems. In *Proceedings of the 2000 IEEE International Conference on Robotics & Automation*, number April, pages 67–72, San Francisco, 2000.

[24] C. Cote, Y. Brosseau, D. Letourneau, C. Ra{\"\i}evsky, and F. Michaud. Robotic Software Integration Using MARIE. *International Journal of Advanced Robotic Systems*, 3(1):55–60, 2006.

[25] A. Eden and T. Mens. Measuring software flexibility. *IEE Proceedings - Software*, 153(3):113–126, 2006.

[26] R. J. Firby. *Adaptive execution in complex dynamic worlds*. PhD thesis, Yale University, 1989.

[27] R. R. Firby. An investigation into reactive planning in complex domains. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, volume 50, pages 202–206, Oct. 1987.

[28] A. Flynn and R. Brooks. Battling reality, 1989.

[29] J. Fritsch, M. Kleinehagenbrock, a. Haasch, S. Wrede, and G. Sagerer. A Flexible Infrastructure for the Development of a Robot Companion with Extensible HRI-Capabilities. *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, (April):3408–3414, 2005.

[30] J. Fritsch and S. Wrede. An Integration Framework for Developing Interactive Robots. In *Software Engineering for Experimental Robotics*, pages 291–305. 2007.

[31] M. Fujita, Y. Kuroki, T. Ishida, and T. T. Doi. Autonomous Behavior Control Architecture of Entertainment Humanoid Robot SDR-4X. In *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, number October, pages 960–967, Las Vegas, 2003.

[32] T. Fukuda and N. Kubota. An intelligent robotic system based on a fuzzy approach. *Proceedings of the IEEE*, 87(9):1448–1470, 1999.

[33] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: elements of reusable object-oriented software*, volume 206. Addison-wesley Reading, MA, 1995.

[34] D. Garlan and M. Shaw. An introduction to software architecture. *World Scientific*, (January), 1994.

[35] E. Gat. Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots. In *Proceedings of the National Conference on Artificial Intelligence*, pages 809—-809, 1992.

[36] E. Gat. On Three-Layer Architectures. *Artificial intelligence and mobile robots: case studies of successful robot systems*, 195:195—-210, 1998.

[37] B. Gates. A robot in every home. *Scientific American*, 296(1):58–65, Jan. 2007.

[38] B. P. Gerkey, R. T. Vaughan, and A. Howard. The Player / Stage Project : Tools for Multi-Robot and Distributed Sensor Systems. In *Proceedings of the International Conference on Advanced Robotics (ICAR 2003)*, pages 317–323, 2003.

[39] D. Glas, S. Satake, T. Kanda, and N. Hagita. An interaction design framework for social robots. *Robotics: Science and Systems VII*, 2012.

[40] D. Goldberg and M. J. Matari. Reward Maximization in a Non-Stationary Mobile Robot Environment. In *The fourth international conference on autonomous agents (agents 2000)*, pages 92–99, 2000.

[41] R. Golombek, S. Wrede, M. Hanheide, and M. Heckmann. Learning a Probabilistic Error Detection Model for Robotic Systems. In *EEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2745–2750, 2010.

[42] D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier. Mechatronic design of NAO humanoid. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, volume 67, pages 769–774, Kobe, Japan, 2009.

[43] V. Graefe. Perception and Situation Assessment for Behavior-Based Robot Control. In *Proc. Int. Conf. on Intelligent Autonomous Systems*, number June, pages 376–383, Sapporo, Japan, 1998.

[44] A. Haasch, S. Hohenner, S. Hüwel, M. Kleinehagenbrock, S. Lang, I. Toptsis, G. A. Fink, J. Fritsch, B. Wrede, and G. Sagerer. Biron–the bielefeld robot companion. In *Proc. Int. Workshop on Advantages in Service Robotics*, number May, pages 27–32, 2004.

[45] M. Hanheide and G. Sagerer. Active memory-based interaction strategies for learning-enabling behaviors. In *RO-MAN 2008 - The 17th IEEE International Symposium on Robot and Human Interactive Communication*, pages 101–106. Ieee, Aug. 2008.

[46] D. Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.

[47] D. Harel. Executable object modeling with statecharts. *Computer*, pages 31–42, 1997.

[48] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. Statemate: A working environment for the development of complex reactive systems. *Software Engineering, IEEE Transactions on*, 16(4):403—-414, 1990.

[49] R. Hartley and F. Pipitone. Experiments with the subsumption architecture. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, number April, pages 1652–1658, 1991.

[50] N. Hawes, M. Zillich, and J. Wyatt. BALT & CAST : Middleware for Cognitive Robotics. *Design*.

[51] R. Hilliard. IEEE-Std-1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems. *IEEE, http://standards. ieee. org*, 2000.

[52] P. Hudak, A. Courtney, H. Nilsson, and J. Peterson. Arrows , Robots , and Functional Reactive Programming. *Advanced Functional Programming*, pages 159–187, 2003.

[53] H. Hüttenrauch and K. S. Eklundh. Fetch-and-carry with CERO : Observations from a long-term user study with a service robot. In *Robot and Human Interactive Communication, 2002. Proceedings. 11th IEEE International Workshop on*, pages 158—-163, 2002.

[54] IEEE. Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990*, 1, 1990.

[55] R. E. Johnson and B. Foote. Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1(2):22–35, 1988.

[56] K. Jüngling, M. Arens, M. Hanheide, and G. Sagerer. Fusion of perceptual processes for real-time object tracking. In *International Conference on Information Fusion*, pages 1–8, 2008.

[57] G. a. Kaminka and I. Frenkel. Integration of Coordination Mechanisms in the BITE Multi-Robot Architecture. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 2859–2866. Ieee, Apr. 2007.

[58] D. Katz, Y. Pyuro, and O. Brock. Learning to Manipulate Articulated Objects in Unstructured Environments Using a Grounded Relational Representation. In *Proceedings of Robotics: Science and Systems IV*, pages 254–261, Zurich, Switzerland, 2008.

[59] C. Kemp. Challenges for robot manipulation in human environments. *Robotics & Automation Magazine, IEEE...*, (March):20–29, 2007.

[60] G. Kiczales. Towards a New Model of Abstraction in the Engineering of Software. In *International Workshop on Reflection and Meta-Level Architecture*, pages 67–76, 1992.

[61] H. Kitano, S. Tadokoro, I. Noda, H. Matsubara, T. Takahashi, A. Shinjou, and S. Shimada. RoboCup Rescue: Search and Rescue in Large-Scale Disasters as a Domain for Autonomous Agents Research. In *Conference on Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings.*, volume 6, pages 739–743, 1999.

[62] M. Kleinehagenbrock. *Interaktive Verhaltenssteuerung für robot companions.* PhD thesis, Bielefeld University, 2004.

[63] S. Kopp, B. Krenn, and S. Marsella. Towards a common framework for multimodal generation: The behavior markup language. In *Intelligent Virtual Agents*, pages 205–217. 2006.

[64] S. Kornblum, T. Hasbroucq, and A. Osman. Dimensional overlap: cognitive basis for stimulus-response compatibility–a model and taxonomy. *Psychological Review*, 97(2):253–270, Apr. 1990.

[65] J. Košeckà and R. Bajcsy. Discrete Event Systems for autonomous mobile agents. *Robotics and Autonomous Systems*, 12(3-4):187–198, Apr. 1994.

[66] P. Kruchten. An Ontology of Architectural Design Decisions in Software-Intensive Systems. In *2nd Groningen Workshop on Software Variability*, pages 54–61, 2004.

[67] I. Kruijff-Korbayová, G. Athanasopoulos, A. Beck, P. Cosi, H. Cuayahuitl, T. Dekens, V. Enescu, A. Hiolle, B. Kiefer, H. Sahli, M. Schröder, G. Sommavilla, F. Tesser, and W. Verhelst. An event-based conversational system for the Nao robot. In *Proceedings of the Paralinguistic Information and its Integration in Spoken Dialogue Systems Workshop*, pages 125–132, 2011.

[68] P. Langley, J. Laird, and S. Rogers. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2):141–160, 2009.

[69] P. A. Laplante. *What Every Engineer Should Know about Software Engineering*. CRC Press, Boca Raton, FL, 2007.

[70] S. Li, M. Kleinehagenbrock, J. Fritsch, B. Wrede, and G. Sagerer. "BIRON, let me show you something": Evaluating the Interaction with a Robot Companion. In *Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics*, number October, pages 2827–2834, 2004.

[71] L.-J. Lin, R. Simmons, and C. Fedor. Experience with a Task Control Architecture for Mobile Robots. Technical Report February, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[72] M. Lindstrom, A. Oreback, and H. Christensen. BERRA: A Research Architecture for Service Robots. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, pages 3278—-3283, 2000.

[73] J. Liu, H. Hu, and D. Gu. A Hybrid Control Architecture for Autonomous Robotic Fish. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, number 1, pages 312–317, Beijing, China, Oct. 2006. Ieee.

[74] M. Lohse. *Investigating the influence of situations and expectations on user behavior: empirical analyses in human-robot interaction*. PhD thesis, 2010.

[75] M. Lohse. Bridging the gap between users' expectations and system evaluations. In *Proceedings of the 20th IEEE International Symposium on Robot and Human Interactive Communication*, pages 485–490, Atlanta, GA, 2011.

[76] M. Lohse and M. Hanheide. Evaluating a social home tour robot applying heuristics. Technical report, 2008.

[77] M. Lohse, M. Hanheide, K. J. Rohlfing, and G. Sagerer. Systemic interaction analysis (SInA) in HRI. In *Proceedings of the 4th ACM/IEEE international conference on Human robot interaction - HRI '09*, page 93, New York, New York, USA, 2009. ACM Press.

[78] I. Lütkebohle, R. Philippsen, V. Pradeep, E. Marder-Eppstein, and S. Wachsmuth. Generic middleware support for coordinating robot software components: The Task-State-Pattern. *Journal of Software Engineering for Robotics*, 2(September):20–39, 2011.

[79] I. Lütkebohle and S. Wachsmuth. Requirements and a Case-Study for SLE from Robotics : Event-oriented Incremental Component Construction. In *Workshop on Software-Language-Engineering for Cyber-Physical Systems*, 2011.

[80] I. Lütkebohle and S. Wachsmuth. Event-oriented Incremental Component Construction. In *Towards Service Robots for Everyday Environments*, pages 445–456. 2012.

[81] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, and R. Metz. Reference Model for Service Oriented Architecture 1.0. Technical Report May, Organization for the Advancement of Structured Information Standards (OASIS), 2005.

[82] D. C. D. MacKenzie, R. C. Arkin, and J. M. J. Cameron. Multiagent mission specification and execution. *Autonomous Robots*, 4(1):29–52, 1997.

[83] P. Maes. How to do the right thing. *Connection Science*, 1(3):291—-323, 1989.

[84] C. Martin, A. Scheidig, and T. Wilhelm. A new control architecture for mobile interaction-robots. In *Proc. of the 2nd European Conference on Mobile Robots (ECMR 2005)*, volume 2005, pages 224–229, 2005.

[85] M. Mataric. Learning in behavior-based multi-robot systems: Policies, models, and other agents. *Journal of Cognitive Systems Research*, 2:81–93, 2001.

[86] D. Meger, P. Forssén, and K. Lai. Curious george: An attentive semantic robot. *Robotics and Autonomous Systems*, 56(6), 2008.

[87] M. Mernik, J. Heering, and A. M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344, Dec. 2005.

[88] G. Metta, P. Fitzpatrick, and L. Natale. YARP: Yet Another Robot Platform. *International Journal of Advanced Robotic Systems*, 3(1):1, 2006.

[89] R. J. Mitchell. *Managing Complexity in Software Engineering*. Peter Peregrinus Ltd., London, 17 edition, 1990.

[90] N. Mohamed, J. Al-Jaroodi, and I. Jawhar. Middleware for robotics: A survey. In *Robotics, Automation and Mechatronics, 2008 IEEE Conference on*, number Ram, pages 736–742, 2008.

[91] V. Mohan and P. Morasso. How past experience, imitation and practice can be combined to swiftly learn to use novel "tools": Insights from skill learning experiments with baby humanoids. In *Biomimetic and Biohybrid Systems, Proceedings of the First International Conference Living Machines*, pages 180–191, Barcelona, Spain, 2012.

[92] M. Montemerlo, N. Roy, and S. Thrun. Perspectives on Standardization in Mobile Robot Programming : The Carnegie Mellon Navigation (CARMEN) Toolkit. In *Intelligent Robots and Systems (IROS 2003). Proceedings of the 2003 IEEE/RSJ International Conference on*, number October, pages 2436–2441, 2003.

[93] M. Montemerlo, S. Thrun, H. Dahlkamp, D. Stavens, and S. Strohband. Winning the DARPA Grand Challenge with an AI robot. *Proceedings of the national conference on artificial intelligence*, 21(1), 2006.

[94] D. Nardi, J.-D. Dessimoz, P. F. Dominey, L. Iocchi, J. Ruiz-del Solar, P. E. Rybski, J. Savage, S. Schiffer, K. Sugiura, T. Wisspeintner, T. van der Zant, and A. Yazdani. RoboCup@Home Rules & Regulations, 2009.

[95] D. Nardi, J.-D. Dessimoz, P. F. Dominey, L. Iocchi, J. Ruiz-del solar, P. E. Rybski, J. Savage, S. Schiffer, K. Sugiura, T. Wisspeintner, T. van der Zant, A. Yazdani, D. Holz, G. Kraetzschmar, D. Gossow, and S. Olufs. RoboCup@Home Rules & Regulations, 2011.

[96] I. a. Nesnas. CLARAty: an architecture for reusable robotic software. *Proceedings of SPIE*, 5083:253–264, 2003.

[97] T. Niemueller, A. Ferrein, D. Beck, and G. Lakemeyer. Design principles of the component-based robot software framework fawkes. In *Simulation, Modeling, and Programming for Autonomous Robots*, pages 300–311, 2010.

[98] N. Nilsson. Shakey the robot. Technical report, SRI International, Menlo Park,CA, 1984.

[99] A. Oreback and H. I. Christensen. Evaluation of Architectures for Mobile Robotics. *Autonomous Robots*, 14(1):33–49, 2003.

[100] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[101] J. Peltason, F. Siepmann, T. Spexard, B. Wrede, M. Hanheide, and E. Topp. Mixed-initiative in human augmented mapping. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2146–2153. IEEE, May 2009.

[102] J. Peltason and B. Wrede. Pamini : A framework for assembling mixed-initiative human-robot interaction from generic interaction patterns. In *SIGDIAL 2010 Conference*, Tokyo, Japan, 2010.

[103] J. Peterson, G. D. Hager, and P. Hudak. A Language for Declarative Robotic Programming. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, number May, pages 1144–1151, 1999.

[104] P. Pirjanian. Behavior coordination mechanisms - State-of-the-art. Technical Report 213, Institute for Robotics and Intelligent Systems, University of Southern California, Los Angeles, 1999.

[105] A. P. Pobil, R. Madhavan, and E. Messina. Lecture Notes for Benchmarks in Robotics Research. Technical report, 2007.

[106] E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier. Choregraphe: a graphical tool for humanoid robot programming. In *Robot and Human Interactive Communication, 2009. RO-MAN 2009. The 18th IEEE International Symposium on*, pages 46–51, 2009.

[107] H. Prendinger, S. Descamps, and M. Ishizuka. MPML: a markup language for controlling the behavior of life-like characters. *Journal of Visual Languages & Computing*, 15(2):183–203, Apr. 2004.

[108] Q. Qiao and P. A. Beling. Behavior Pattern Recognition using A New Representation Model. *Computing Research Repositiory (CoRR)*, abs/1301.3:1–11, Jan. 2013.

[109] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, volume 3, 2009.

[110] J. Rosenblatt and D. Payton. A fine-grained alternative to the subsumption architecture for mobile robot control. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, volume 2, pages 317–323. Ieee, 1989.

[111] A. Safiotti. The Uses of Fuzzy Logic in Autonomous Robot Navigation: a catalogue raisonne. In *Fuzzy Systems, 1997., Proceedings of the Sixth IEEE International Conference on*, pages 573–578, 1997.

[112] S. Saint-Aime, B. Le-Pevedic, D. Duhaut, and T. Shibata. EmotiRob: Companion robot Project. In *RO-MAN 2007 - The 16th IEEE International Symposium on Robot and Human Interactive Communication*, pages 919–924. IEEE, 2007.

[113] F. B. Schneider. The State Machine Approach : A Tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299—-319, 1990.

[114] S. Schneider. Integration einer humanoiden Robotikplattform in eine serviceorientierte Architektur, 2010.

[115] F. Schubert, T. Spexard, M. Hanheide, and S. Wachsmuth. Active Vision-based Localization For Robots In A Home-Tour Scenario. In *The 5th International Conference on Computer Vision Systems*, number Icvs, 2007.

[116] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake. Real-time human pose recognition in parts from single depth images. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1297–1304. Ieee, June 2011.

[117] R. Siegwart and K. Arras. Robox at Expo.02: A large-scale installation of personal robots. *Robotics and Autonomous Systems*, 42:203–222, 2003.

[118] F. Siepmann and S. Wachsmuth. A Modeling Framework for Reusable Social Behavior. In R. De Silva and D. Reidsma, editors, *Work in Progress Workshop Proceedings ICSR 2011*, pages 93–96, Amsterdam, 2011. Springer.

[119] F. Siepmann, L. Ziegler, M. Kortkamp, and S. Wachsmuth. Deploying a modeling framework for reusable robot behavior to enable informed strategies for domestic service robots. *Robotics and Autonomous Systems*, Nov. 2012.

[120] R. Simmons and D. Apfelbaum. A task description language for robot control. In *Proceedings of the International Conference on Intelligent Robots and Systems*, volume 3, pages 1931–1937. Ieee, 1998.

[121] C. M. U. Software Engineering Institute. Software Architecture Glossary.

[122] T. Spexard and M. Hanheide. System Integration Supporting Evolutionary Development and Design. In *Human Centered Robot Systems*, pages 1–9, 2009.

[123] T. Spexard, M. Hanheide, S. Li, and B. Wrede. Oops, Something Is Wrong - Error Detection and Recovery for Advanced Human-Robot-Interaction. In *Proc. of the Workshop on Social Interaction with Intelligent Indoor Robots at the Int. Conf. on Robotics and Automation*, 2008.

[124] T. P. Spexard, F. H. Siepmann, and G. Sagerer. Memory-based Software Integration for Development in Autonomous Robotics. In *Intelligent Autonomous Systems 10 (IAS 10)*, page 49. Ios Pr Inc, 2008.

[125] J. Stolarz and P. Rybski. An architecture for the rapid development of robot behaviors. *Robotics*, (May), 2007.

[126] J. Stuckler and S. Behnke. Benchmarking mobile manipulation in everyday environments. *2012 IEEE Workshop on Advanced Robotics and its Social Impacts (ARSO)*, (May):1–6, May 2012.

[127] S. Thrun and M. Beetz. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *The International Journal of Robotics Research*, 19(11):972–999, 2000.

[128] C.-Y. Tseng, Y.-L. Huang, and J.-S. Hu. ESAIR: A Behavior-Based Robotic Software Architecture on Multi-Core Processor Platforms. *International Journal of Automation and Smart Technology*, 3(1):47–56, Mar. 2013.

[129] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, J. Dolan, D. Duggins, D. Ferguson, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. Howard, A. Kelly, D. Kohanbash, M. Likhachev, N. Miller, K. Peterson, R. Rajkumar, P. Rybski, B. Salesky, S. Scherer, Y. Woo-Seo, R. Simmons, S. Singh, J. Snider, A. Stentz, W. Whittaker, J. Zigla, H. Bae, B. Litkouhi, J. Nickolaou, V. Sadekar, S. Zeng, G. Motors, Caterpillar, and C. AG. Tartan Racing: A Multi-Modal Approach to the DARPA Urban Challenge. Technical report, 2007.

[130] T. van der Zant and T. Wisspeintner. Robocup@ home: Creating and benchmarking tomorrows service robot applications. *Robotic Soccer*, (December):521–528, 2007.

[131] S. Wachsmuth, M. Hanheide, F. Siepmann, and T. Spexard. ToBI-Team of Bielefeld: The Human-Robot Interaction System for RoboCup@ Home 2009. Technical report, Graz, Austria, 2009.

[132] S. Wachsmuth, F. Siepmann, D. Schulze, and A. Swadzba. ToBI - Team of Bielefeld : The Human-Robot Interaction System for RoboCup @ Home 2010. Technical report, Singapore, 2010.

[133] S. Wachsmuth, F. Siepmann, L. Ziegler, and F. Lier. ToBI - Team of Bielefeld : The Human-Robot Interaction System for RoboCup @ Home 2011. Technical report, Istanbul, Turkey, 2011.

[134] S. Wachsmuth, F. Siepmann, L. Ziegler, F. Lier, and M. Schöpfer. ToBI-Team of Bielefeld: The Human-Robot Interaction System for RoboCup@HOME 2012. Technical report, Mexico City, Mexico, 2012.

[135] S. Wachsmuth, S. Wrede, and M. Hanheide. Coordinating interactive vision behaviors for cognitive assistance. *Computer Vision and Image Understanding*, 108:135—-149, 2007.

[136] M. L. Walters, K. Dautenhahn, S. N. Woods, K. L. Koay, C. Lane, and H. Uk. Robotic Etiquette : Results from User Studies Involving a Fetch and Carry Task. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, pages 317–324, 2007.

[137] D. Westhoff, T. Scherer, H. Stanek, J. Zhang, A. Knoll, and T. U. Munich. A flexible framework for task-oriented programming of service robots. *VDI Berichte*, 1841:737—-744, 2004.

[138] J. Wienke and S. Wrede. A middleware for collaborative research in experimental robotics. *2011 IEEE/SICE International Symposium on System Integration (SII)*, pages 1183–1190, Dec. 2011.

[139] M. Wirsing. Software Intensive Systems. *Report on ERCIM Beyond the Horizon Thematic Group 6*, 6(June):1–42, 2006.

[140] T. Wisspeintner, T. van Der Zan, L. Iocchi, S. Schiffer, and T. V. D. Zan. RoboCup@ Home: Results in Benchmarking Domestic Service Robots. *RoboCup 2009: Robot Soccer World Cup XIII*, pages 390–401, 2010.

[141] T. Wisspeintner, T. van der Zant, L. Iocchi, and S. Schiffer. RoboCup@Home: Scientific Competition and Benchmarking for Domestic Service Robots. *Interaction Studies*, 10(3):392–426, Dec. 2009.

[142] B. G. Woolley and G. L. Peterson. Unified Behavior Framework for Reactive Robot Control. *Journal of Intelligent and Robotic Systems*, 55(2-3):155–176, Dec. 2008.

[143] B. Wrede, A. Haasch, N. Hofemann, S. Hohenner, S. Hüwel, M. Kleinehagenbrock, S. Lang, S. Li, I. Toptsis, G. A. G. Fink, J. Fritsch, and G. Sagerer. Research issues for designing robot companions: BIRON as a case study. In *Proc. IEEE Conf. on Mechatronics & Robotics*, volume 4, pages 1491—-1496, 2004.

[144] S. Wrede. *An information-driven architecture for cognitive systems research.* PhD thesis, 2008.

[145] Z. Xue, S. Ruehl, A. Hermann, T. Kerscher, and R. Dillmann. An autonomous ice-cream serving robot. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3451—-3452, 2011.

[146] B. Yamauchi. A frontier-based approach for autonomous exploration. In *Computational Intelligence in Robotics and Automation Proceedings IEEE International Symposium on*, pages 146–151. IEEE Comput. Soc. Press, 1997.

[147] M. Yannakakis. Hierarchical state machines. In *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics*, pages 315–330. 2000.

[148] C. Ye. Development of a 3D Snake-like Robot :. In *Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation*, pages 117–122, Harbin, China, 2007.

[149] L. Ziegler, F. Siepmann, M. Kortkamp, and S. Wachsmuth. Towards an Informed Search Behavior for Domestic Robots. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 1–10, 2010.