Comparison of Behaviour-Based Architectures for a Collaborative Package Delivery Task

Melanie Remmels, Nicolás Navarro-Guerrero and Stefan Wermter
Universität Hamburg, Department of Informatics, Knowledge Technology
Vogt-Kölln-Str. 30, 22527 Hamburg, Germany
nicolas.navarro.guerrero@gmail.com, wermter@informatik.uni-hamburg.de

ABSTRACT
A comparison between behavioural architectures, specifically a BDI architecture and a finite-state machine, for a collaborative package delivery system is presented. The system should assist a user in handling packages in cluttered environments. The entire system is built using open-source solutions for modules including speech recognition, person detection and tracking, and navigation. For the comparison, we use three criteria, namely, a static implementation-based comparison, a dynamic comparison and a qualitative comparison. Based on our results, we provide experimental evidence that supports the theoretical consensus about the domain of applicability for both, BDI architectures and finite-state machines. However, we cannot support or discourage any of the tested architectures for the particular case of the collaborative package delivery scenario, due to the non-overlapping strengths and weakness of both approaches. Finally, we outline future improvements to the system itself as well as the comparison of both behavioural architectures.

ACM Classification Keywords
H.3.4. Systems and Software – Performance evaluation (efficiency and effectiveness); I.2.9. Robotics – Autonomous vehicles

Author Keywords
Behaviour-Based Architectures; Human-Robot Collaboration; Package Delivery Scenario

INTRODUCTION
The interest in fully or semi-automated package delivery is growing. Large companies are investing in solutions to automate the whole chain of package delivery. This includes Amazon [5, 2, 28], DHL in collaboration with Effistore [10] and Hermes in collaboration with Starship Technologies [35, 33, 19]. A similar trend is also arising from the consumer market. For instance, Piaggio [26] is working on the semi-autonomous Gita robot. Gita is capable of following a walking human and carrying up to 18 kg of groceries, and further capabilities are being developed [27].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

HAI'17, October 17–20, 2017, Bielefeld, Germany

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-5113-3/17/10…$15.00

DOI: **https://doi.org/10.1145/3125739.3125764**

One of the main limitations of the mentioned industrial and personal systems is that they are tailored solutions for warehouses or outdoor environments. However, the possibility for package delivery to operate in narrow and cluttered indoor environments such as offices or the challenges of tight human-robot collaboration in such environments are not explicitly described. Among these challenges is the effective coordination of behaviour. A variety of behaviour architectures exists in different areas of specialisation, namely reactive approaches, deliberative approaches, hybrid approaches and behaviour-based control [24]. From these classes, we see deliberative and behaviour-based control strategies as most suitable for a collaborative package delivery robot due to their flexibility and availability of control frameworks.

From a theoretical point of view, it is argued that deliberative behaviour approaches are more appropriate for large and complex use cases. Further, deliberative approaches provide an abstraction layer where only the tasks are specified but not their order of execution [16]. On the other hand, state machines should be used for small and simple uses cases. State machines are particularly suitable for static action sequences with high demand for repeatability [16]. This abstract and high-level differentiation can provide a hint to which approach is more suitable for a given use case. Unfortunately, there are no guidelines to decide on the complexity and size of a use case and thus the question of which approach is more suitable for a collaborative package delivery scenario arises.

In this paper, we compare both behaviour architectures, the deliberative approach and the state machine, for a collaborative package delivery task, taking into consideration performance as well as their difficulties of application in such a scenario. The whole use case is implemented by using open-source tools. We then compare them considering three metrics including static, dynamic and qualitative variables. Finally, we report and discuss the results outlining future lines of research.

BEHAVIOUR ARCHITECTURES
The use case of the collaborative package delivery needs concurrent processing, like navigation and speech commands. To handle this parallelism, a behaviour architecture is needed

which interacts and controls the system. The architecture can be modelled in a fixed, predefined way or leave some autonomy to the system which can find different ways to solve the task. The choice of which frameworks are compared has been done after two criteria. The approaches should be as different as possible so that both conceptual and implementation-related differences can be made visible. If someone deals with behavioural architectures in the field of robotics, one often encounters two concepts: Planning and State machines. A further argument for the selection of the behavioural architectures to be compared was the compatibility with ROS. The used hardware provides integration with ROS. Therefore, it should be possible to integrate the behavioural architectures into ROS, the de facto standard for robotics control software.

Deliberative Approaches

Deliberative approaches combine all available sensory information with their internal knowledge of the environment. This information is used to decide on the next action. The decision which action to choose is done by searching possible state-action sequences [24].

This search can be split into three phases: Firstly, the input to the system is obtained by the sensors. Secondly, this input is analysed and used for planning the next steps of the system. Finally, the system executes the plan. This basic concept is known under different names in different fields, for instance, *Sense-Plan-Act* or *Belief-Desire-Intention (BDI)*. If the action decision is goal-oriented, the concept is named *Planning*. Different implementations exist with variations mostly in the decision step (cf. second step).

The main advantage of deliberative approaches is that there is no need to predefine the number of states or the transition between states to fulfil a task. All of which is decided internally by a planner. This makes deliberative approaches very flexible and scalable [1]. The complexity of the number of plans and the number of available behaviours is linear [4, 16].

One disadvantage of deliberative approaches concerns safety. Because although the transition between states (action sequence) is deterministic, the action sequence is not known to the designer. The fact that the transitions are decided internally also makes the correction of potentially dangerous behaviours difficult [16].

PROFETA

The framework PROFETA (Python RObotic Framework for dEsigning sTrAtegies) is a Python implementation of AgentSpeak [11]. AgentSpeak(L) [11] is one of the used formal languages for the design of a BDI agent. The idea behind it has already been presented by Rao [31].

To use this abstract language, an interpreter has to be designed. The speciality of the PROFETA implementation is, that all parts of the software are written in Python. In other implementations like Jason, a Java implementation of AgentSpeak, the reasoning part is written in the AgentSpeak syntax whereas the actions are implemented in Java. The logic of BDI systems is split into three main types inside PROFETA: Beliefs, Goals and Actions. Each of these types is implemented as a class. All beliefs, goals and actions needed for the implementation are subclasses of the corresponding main classes.

PROFETA is advantageous because it is fully implemented in Python. This means it can be integrated efficiently with existing codes. Formal planning languages, like Strips, PDDL and AgentSpeak, have their own syntax. Their syntax is abstract and is very different from programming languages. So the integration of a planning module into an existing software system is challenging. The AgentSpeak syntax has been adapted for PROFETA but it still differs heavily from common programming languages syntax.

Behaviour-Based Control

In some situations, a complete system has to achieve a specific behaviour. If this system contains distributed modules which have to interact, a behaviour-based control system is used. Each of the modules receives input and reacts to it. The input can either be sensory-based or the result of another behaviour. The reaction to this input can be a command for one of the robot's actuators or the input for another behaviour. The different behaviours can contain states and abstract representations which enable the possibility of hierarchically structured behaviours.

State machines, on the other hand, are more suitable for static systems and for situations where actions sequences can be predefined and need to be repeatable [16]. To build a state machine, all possible states have to be defined explicitly [1, 16]. This leads to the main drawback of the concept, i.e. the complexity of a state machine grows exponentially with the number of transitions [16]. One of the advantages of state machines is their diversity in the application context. This means, many programmers have used them already and they can reuse them for a new context [14]. This knowledge of the concept can explain why design changes have been applied to a state machine 4.5 times faster than to the corresponding BDI model [4]. Regarding safety, interactions completely rely on the considerations of the designer and thus are inherently limited. However, once a certain safety-critical condition is identified, it can be solved knowing the impact on other plans or states.

SMACH

If different steps are needed to reach a target, a directed graph can be used. During the states, the subtask is performed and when one task is finished, the transition to the next state is used. This directed graph is the basic form of a finite state machine [42]. If one subtask involves more complex decisions, it can be represented via an own state machine. This nesting leads to a hierarchical state machine. Each state of a hierarchical state machine can contain a state or another finite state machine. The usage of a hierarchical structure can facilitate the modelling of a large system [42].

An available state machine implementation for ROS (Robotic Operating System [30]) is SMACH (*S*tate *MACH*ine). It has sophisticated features which facilitate the integration with ROS, like specific states for the different communication mechanisms. SMACH is implemented in Python, a programming language with a tight connection to ROS [7, 8].

Synopsis

Deliberative systems provide a level of freedom to the system because the order of actions to reach a target is not predefined. The approach of PROFETA was chosen because it is implemented in Python, a language which is well linked to ROS. For the behaviour-based approaches, a state machine implementation has been chosen because the concept is often used in robotics. SMACH is tightly integrated into ROS and it has already been used in different robotic contexts [3, 41, 36].

SCENARIO AND SYSTEM DESIGN

As described in the introduction, package delivery is a current topic of interest, so it was chosen for the use case of this comparison. Our robot system consists of a Jackal UGV (wheeled mobile platform), a laser-range sensor for navigation and mapping, a Nao for interaction, an Xtion (RGB-D camera) for person detection and tracking, and a headset for speech recognition. The hardware is shown in Figure 1.

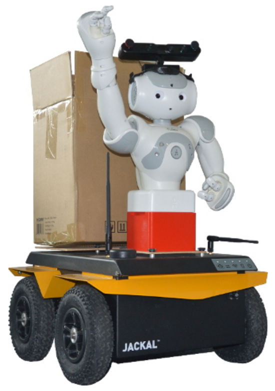

Figure 1. Used hardware of the package delivery system.

The different hardware components interact with each other via the Robotic Operating System (ROS) [30]. Each component is translated to a dedicated software module which can be seen in Figure 2. This centralised architecture facilitates the comparison of different behaviour architectures because the interaction of the behavioural architectures with the other system components is limited to a single module called 'robot'. The 'robot' module controls all interaction with the robotic hardware and other software modules through ROS. This module acts as an abstraction layer for the ROS-related communication. The use case and the task specific modules are described in the sections below.

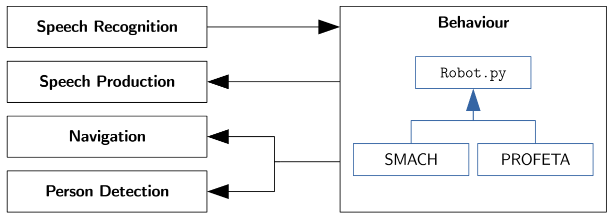

Figure 2. A graphical representation of the architecture of the collaborative package delivery system.

Use Case

The system was created to support the human by handling heavy packages. To achieve this, its capabilities have to be selected in the right order. One possible sequence can be seen in Figure 3. However, alternative behaviours can easily be created by combining the modules in different sequences. For example, after the unloading phase, the system can follow the user to another picking location. To stop the following behaviour, the user uses the command *WAIT*. In this *WAIT*-State, the system can be turned, directed through a door or its loading state can be changed by a command. In all cases, the robot is instructed via speech commands.

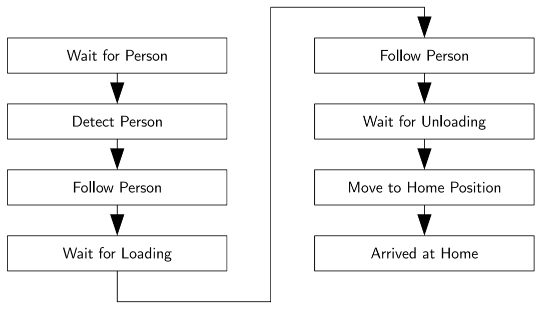

Figure 3. Sequence diagram of the basic use case.

Other functionalities to improve the usability of the system have also been developed but are not part of the tested main use case. For instance, the commands *TURN* and *DOOR* can be used for easier (un)loading of packages and instruct the robot to move first while the user in keeping a door open respectively.

Human-Robot-Interaction

For the interaction between a robot and a human, different channels exist [17]. For proximate interaction, speech, gestures, haptics, actions in the environment or physiological signals, like muscle activities, can be used as possible communication mediums.

In the current implementation, we chose speech due to its intuitive and natural way of interaction [9]. We used DOCKS (DOmain- and Cloud-based Knowledge for Speech recognition) [38] in combination with a simple language understanding model [41] for processing the robot commands.

Person Detection

The detection of a person is crucial in our scenario. Although multiple alternatives exist, for instance, based on skin colour [34], face detection [40], or laser range pattern-based approaches [15], they cannot be used for collaborative package delivery in cluttered office environments because most of the time the user is not facing the robot and pattern-based approaches generate too many false positives in a cluttered office environment. Thus, we use the openni2-tracker [18] to detect and track the user. The openni2-tracker is a skeleton tracker which provides the joint positions of the skeleton model relative to the camera frame. This type of tracker is ideal for our scenario as it is robust to changing lighting conditions, user pose and temporary occlusion.

Navigation

We used the *Adaptive Monte Carlo Localization (AMCL)*-module of ROS, which is based on the work of Thrun et al. [37]. This navigation module operates on laser range sensor data. The *AMCL* provides a current pose estimation of the robot on a given map which together with a path planner enables the robot to autonomously navigate to a point within a known map.

Our high-level navigation module consists of four different ROS services. One for turning at a predefined angle, one for the navigation through a door, one for user following and one to autonomously navigate back to the home position. Each of these services can be activated by voice commands.

COMPARISON OF BEHAVIOURAL ARCHITECTURES

The theoretical description of behaviour architectures provides a hint to which architecture should be used for a given use case. However, none of the criteria of size or complexity has been quantified yet, which raises the following questions: How can we analyse and compare these architectures? How can the different approaches be evaluated?

Adams et al. [1] did a comparison of two behavioural architectures based on the characteristics presented by Mandes and Winker [23]. Their analysis compared a state machine and a planning module in a simulated system. They performed their analysis on a modelling of the decision-making of humans in the context of their escape behaviour in cases of bushfires. Their work resulted in confirming that comparisons of these kinds are needed. The comparison of Adam et al. [1] was based on a simulated system. Although their system was complex, they were able to control and supervise every parameter of the system. Their comparison was only based on three parameters, and they did not provide detailed information how they obtained the results. Additionally, if this comparison is now transferred to a real life situation, the inputs will change. In a real world environment, a significant number of parameters cannot be controlled. Since the inputs depend partially on sensors which can have errors in their readings or a delay through pre- or post-processing. This means that the results reported by Adams et al. [1] have to be verified before the results can be transferred to a human-robot interaction scenario. Thus, we argue that further comparisons are needed which would be tested on real working systems and which could provide a better inside into the comparison itself.

We compare PROFETA (an implementation of a deliberative approach) and SMACH (an implementation for finite hierarchical state machines) in a collaborative package delivery scenario. We follow a centralised control architecture, which, among other benefits, simplifies the comparison of different behaviour architectures. For this, we use an abstraction module that provides all sensor readings and performs all actions that the different behavioural architectures need, see Figure 2.

Particularly, we use three types of metrics which were chosen based on general measures of complexity of a system [22], the computer science tailored version of such general measures of complexity [23] and more specific measures relevant for behaviour-based architectures [1].

The first comparison named *difficulty of description* [22] is a static analysis which considers properties such as the number of parameters, the cyclomatic complexity number and implementation relevant properties [23]. The number of parameters is only relevant for non-linear systems and the cyclomatic complexity number provides an indication of all distinct execution path of a given framework, both metrics are not applicable to our use case and therefore have not been analysed. We concentrate our analysis in implementation-relevant parameters including structure (classes and files), the number of class types, lines of code and indentation (nesting).

The second comparison named *difficulty of creation* [22] is a dynamical analysis. Unfortunately, there is no universal definition of the criterion. Thus, we focus on computational time and memory usage as similarly as done by Adam et al. [1]. Finally, the third comparison presented in this paper considers qualitative properties such as the documentation of the frameworks. Not all parameters and criteria of a framework are measurable by analysing the implementation of a use case. Many factors need to be considered before a framework can be evaluated. Some of these factors are hard to estimate accurately, like the size of the existing community that applies and develops a framework. However, these factors are relevant because they have a direct impact on the speed of development of such a system as well as on the quality of such system. Therefore, here we also present such a qualitative comparison.

Implementation-Based Comparison

For the code-based implementation, it is important that the general file structure is similar for all files. To achieve this, some preprocessing steps have been performed for all files. Specifically, all code related to the behavioural architectures is placed into a single file and all comments and empty lines have been removed.

For the analyses of the code quality, the Python Style Guide [39] was used. After a standardisation of the formatting [20], the quality was assessed by using the tool Pylint [29]. SMACH scored $6.12/10$ while PROFETA scored $-1.33/10$. The big difference between the Pylint scores can be explained by the additional syntax of AgentSpeak [12] which is the reasoning engine behind PROFETA.

Table 1 summarises the score of all parameters compared, which are based on the parameters presented by Mandes and Winker [23] and Adam et al. [1]. The results in Table 1 shows that PROFETA is less complex than SMACH. Only in the category of class types, SMACH achieves better values. The generally higher values of SMACH can be explained by its concept which leads to a higher number of classes but of the same type. SMACH also shows a higher level of nesting which is an indication of code complexity. However, in this case, we attribute it mainly to the syntax of Python and not necessarily to code complexity.

	SMACH	PROFETA
Structure		
Number of files	1	1
Number of classes	7	28
Number of characters	8465	5794
Classes		
Types of Classes	1	4
Distribution of Class types	7 states	6 Beliefs, 10 Goals, 11 Actions, 1 Sensor
Lines of Code		
Total Number of Lines	252	188
Average Line Length	48	36
Maximum Line Length	138	92
Indentation		
Average nested block depth level	3	1
Maximum nested block depth level	11	6

Table 1. Results of the Static Comparison of the Behavioural Architectures

Dynamic Comparison

For the *difficulty of creation*, the computation time and the memory usage were analysed [23]. To avoid stochastic variability typical for such user-dependent systems, we opted for simulating all interactions. Using a simulation for this comparison effectively isolates the components related to the actual behavioural architecture from any uncontrolled source of variability. This allowed as to analyse the desired behaviour and not the error handling mechanisms. Finally, to test the behavioural architecture in action we predefined an interaction sequence that invokes all capabilities of the system at least once, see Table 2. The results of this test are summarised in Table 3, Figure 4 and Figure 5.

1.	DETECT	5.	FOLLOW	9.	WAIT
2.	FOLLOW	6.	WAIT	10.	TURN
3.	WAIT	7.	LOAD	11.	UNLOAD
4.	DOOR	8.	FOLLOW	12.	RETURN

Table 2. Sequence of commands for the system-level tests.

Both behavioural architectures present comparable dynamic performance, without significant differences in computational time, CPU usage or Virtual memory usage. The only larger difference is found for real memory usage where SMACH consistently uses 37% more. SMACH has a shutting down phase which causes a decrease in memory usage at the end of every run, see Figure 4. PROFETA does not have such phase. Overall, these results are in line with the results reported by Adam et al. [1].

	SMACH	PROFETA
Average Time	223.3923 s	222.6165 s
Variance	0.0683	0.0071

Table 3. Average results for the computation time needed for ten repetitions of interaction sequence shown in Figure 2.

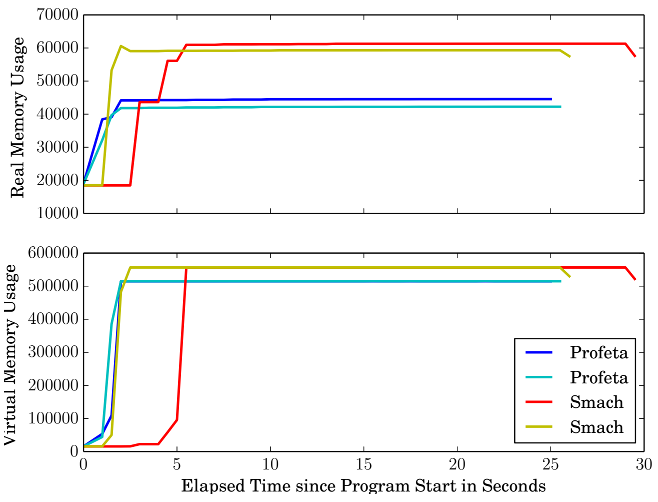

Figure 4. Memory usage of SMACH and PROFETA. SMACH has a shutting down phase which explains the drop in memory usage at the end of every test run.

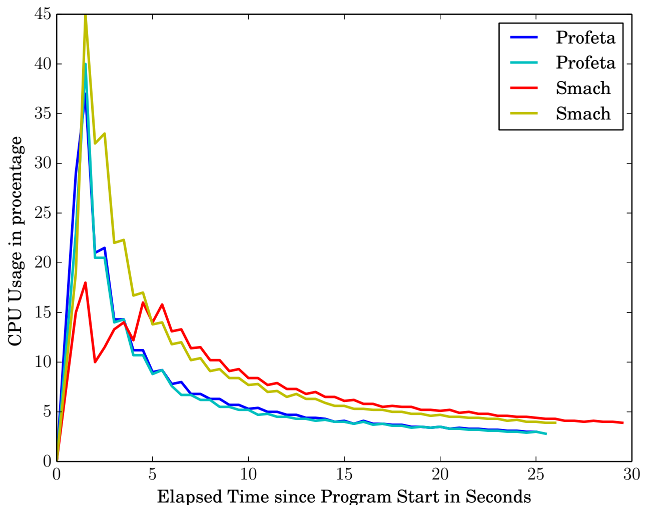

Figure 5. CPU usage of SMACH and PROFETA.

Qualitative Comparison

Table 4 shows diverse qualitative criteria also considered in this work. Qualitatively, SMACH scores better than PRO-

	SMACH	PROFETA
Repository		
Repository	[6]	[32]
Developers/Contributors	14	1
First Commit	28. July 2010	20. November 2012
Last Commit	8. June 2017	22. October 2016
Number of Commits	151	38
Adapting the Code		
First entry (create first Use Case)	easy (tutorials are available)	medium (only few examples are available)
Modify an existing Use Case	hard (all states have to be checked)	easy (one plan has to be added)
Adapt an Use Case slightly	easy (the place of the adaptation is well defined)	hard (action sequences are not expose to the designer)
Number of Publications		
Publications describing the Framework	2 [7, 8]	2 [11, 12]
Authors which described the Framework	10	4
Publications using the Framework	5 [3, 14, 21, 25, 36]	1 [13]
Other		
Visualisation	available	None
Documentation	available	brief overview
Tutorials	available	only some examples
Community	connected to the ROS community	None

Table 4. Comparison of subjective parameters of the Behavioural Architectures

FETA. It has a larger community, a higher acceptance and usage in the robotics community. One of the main disadvantages of PROFETA is the absence of detailed and comprehensive documentation. Some of its functionalities are only briefly mentioned. Consequently, not all its capabilities can be used easily.

Analysis of the Use Case

Although we did not perform a formal user study of the system, we tested the overall functionality of the system as described in Figure 2 with real users. This allowed us to estimate the robustness and error-proneness of the system. The users were provided with a list of sentences which represented the different commands. The sentences are shown in Figure 5. Overall, the system performed well in this small number of trials. It detected the person without any problems and understood the different commands efficiently using a headset.

DETECT	Detect me, Robot.
FOLLOW	Robot, please follow me.
DOOR	Move straight through the door.
WAIT	Robot, please wait.
TURN	Turn around.
LOAD	I am going to load you
UNLOAD	We reached our goal, I am unloading you
RETURN	You can return home
STOP	Stop!

Table 5. Sequence of commands for the system-level tests.

DISCUSSION

We presented a comparison of two behavioural architectures for a collaborative package delivery system. The system should assist a user to handle heavy packages. An area of application could be, for example, an autonomous shopping trolley, a mobile bookshelf for libraries, or basically any application where objects need to be moved around while following a person.

The collaborative package delivery system is built by combining existing solutions for the needed capabilities. It consists of four main modules. For the human-robot interaction, the speech recognition system DOCKS is used [38]. To track a person, a skeleton-based approach is used [18]. To follow a person, the values of the person detection are converted into map coordinates which are used for the navigation of the robot which uses the AMCL-approach by ROS [37]. The behaviour module was implemented with two different frameworks: a state machine-based approach named SMACH [7, 8] and a BDI-approach named PROFETA [11, 13]. Both frameworks implement the same behaviour and were compared in this work.

We performed three comparisons, namely, a static implementation-based comparison, a dynamic comparison and a qualitative comparison. Our implementation-based comparison shows that PROFETA requires less code than SMACH for the implementation of the same behaviour. The dynamic comparison does not show any major differences between the frameworks in terms of computational time, CPU usage or virtual memory usage. We only found a significant difference in the amount of real memory usage where SMACH requires 37% more memory. Finally, for the qualitative comparison, we argue that SMACH scores higher because it is a more mature framework and has a larger user base than PROFETA.

Overall, we agree with the theoretical consensus that the BDI concept is better for larger use cases, however, we also argue that a more programming-friendly framework is required. Particularly for the collaborative package delivery scenario, it is difficult to argue against or in favour of any of the tested frameworks, because although PROFETA scores better in our static and partially better in the dynamic comparison, it lacks the required documentation and technical support. On the other hand, although SMACH requires a longer and more complex code, this can be overcome by the available documentation, examples and supporting community.

Finally, as future work, we would like to improve the implemented systems as well as to further compare different aspects of both behavioural architectures. From the system itself, speech recognition is the most important module that needs improvements. It would be desirable to allow the user to instruct the system using natural speech instead of predefined commands. The person detection could be extended by identifying the current user which could allow the system to be used with multiple people in the field of view. The navigation module needs further development to cope with cluttered and narrow spaces. After the improvements in all these modules are completed, a user study would be required to assess the usefulness of the system and further refine the use case.

Regarding the comparison of behavioural architectures, it would be interesting to quantify the size and complexity of use cases to facilitate the choice of a behavioural architecture. For a further validation of our qualitative comparison, a developer-oriented user study would be required. In this user study aspects such as the difficulty of appropriation [1], i.e. the difficulty of extending or modifying an existing use case should be evaluated by letting developers outside the project try to understand and modify the code.

ACKNOWLEDGEMENTS
The authors gratefully acknowledge partial support from the German Research Foundation DFG under project CML (TRR 169), the European Union under project SECURE (No 642667), and the Hamburg Landesforschungsförderungsprojekt CROSS.

REFERENCES

1. Carole Adam, Patrick Taillandier, and Julie Dugdale. 2017. Comparing Agent Architectures in Social Simulation: BDI Agents Versus Finite-State Machines. In *Hawaii International Conference on System Sciences (HICSS)*. ScholarSpace, Hilton Waikoloa Village, HI, USA, 267–273. http://hdl.handle.net/10125/41181
2. Amazon. 2015. Amazon Prime Air. (2015). https://www.youtube.com/watch?v=MXo_d6tNWuY
3. Markus Bajones, Daniel Wolf, Johann Prankl, and Markus Vincze. 2014. Where to Look First? Behaviour Control for Fetch-and-Carry Missions of Service Robots. In *Austrian Robotics Workshop (ARW)*. Linz, Austria, 115–120. http://www.roboticsworkshop.at/index.php/previous-workshops/arw-2014
4. Arran Bartish and Charles Thevathayan. 2002. BDI Agents for Game Development. In *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Vol. 2. ACM, Bologna, Italy, 668–669. DOI:http://dx.doi.org/10.1145/544862.544901
5. Greg Bensinger. 2015. Amazon Touts New Drone Prototype, Plans Multiple Designs. *Digits - The Wall Street Journal* (2015). https://blogs.wsj.com/digits/2015/11/29/amazon-touts-new-drone-prototype-plans-multiple-designs/
6. Jonathan Bohren. 2014. Ros/Executive_smach Repository. (2014). https://github.com/ros/executive_smach License: BSD http://wiki.ros.org/smach.
7. Jonathan Bohren and Steve Cousins. 2010. The SMACH High-Level Executive. *IEEE Robotics Automation Magazine* 17, 4 (2010), 18–20. DOI: http://dx.doi.org/10.1109/MRA.2010.938836
8. Jonathan Bohren, Radu Bogdan Rusu, E. Gil Jones, Eitan Marder-Eppstein, Caroline Pantofaru, Melonee Wise, Lorenz Mösenlechner, Wim Meeussen, and Stefan Holzer. 2011. Towards Autonomous Robotic Butlers: Lessons Learned with the PR2. In *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, Shanghai, China, 5568–5575. DOI: http://dx.doi.org/10.1109/ICRA.2011.5980058
9. Jan Peter de Ruiter. 2004. On the Primacy of Language in Multimodal Communication. In *International Conference on Language Resources and Evaluation (LREC) (Workshop on Multimodal Corpora: Models of Human Behaviour for the Specification and Evaluation of Multimodal Input and Output Interfaces)*. European Language Resources Association (ELRA), Lisbon, Portugal, 38–41.
10. Effidence. 2017. Official Website of Effidence. (2017). http://www.effidence.com/
11. Loris Fichera, Daniele Marletta, Vincenzo Nicosia, and Corrado Santoro. 2010a. Flexible Robot Strategy Design Using Belief-Desire-Intention Model. In *Research and Education in Robotics - EUROBOT 2010: International Conference, Rapperswil-Jona, Switzerland, May 27-30, 2010, Revised Selected Papers*. Number 156 in Communications in Computer and Information Science. Springer Berlin Heidelberg, 57–71. DOI: http://dx.doi.org/10.1007/978-3-642-27272-1_5
12. Loris Fichera, Daniele Marletta, Vincenzo Nicosia, and Corrado Santoro. 2010b. A Methodology to Extend Imperative Languages with AgentSpeak Declarative Constructs. In *Workshop Nazionale "Dagli Oggetti Agli Agenti" (WOA) (11)*, Vol. 621. CEUR-WS.org, Rimini, Italy. http://woa10.apice.unibo.it/
13. Giancarlo Fortino, Wilma Russo, and Corrado Santoro. 2013. Translating Statecharts-Based into BDI Agents: The DSC/PROFETA Case. In *Multiagent System Technologies (LNCS)*, Vol. 8076. Springer, Berlin, Heidelberg, Koblenz, Germany, 264–277. DOI: http://dx.doi.org/10.1007/978-3-642-40776-5_23
14. Michalis Foukarakis, Asterios Leonidis, Margherita Antona, and Constantine Stephanidis. 2014. Combining Finite State Machine and Decision-Making Tools for Adaptable Robot Behavior. In *Universal Access in Human-Computer Interaction. Aging and Assistive Environments (LNCS)*, Vol. 8515. Springer, Cham, Crete, Greece, 625–635. DOI: http://dx.doi.org/10.1007/978-3-319-07446-7_60

15. Rachel Gockley, Jodi Forlizzi, and Reid Simmons. 2007. Natural Person-Following Behavior for Social Robots. In *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Vol. 2nd. IEEE, Arlington, VA, USA, 17–24. DOI:http://dx.doi.org/10.1145/1228716.1228720

16. Mario Gongora and David Irvine. 2010. Adaptive Intelligent Agents Based on Efficient Behaviour Differentiation Models. In *IEEE ANDESCON*. IEEE, Bogota, Colombia, 1–6. DOI: http://dx.doi.org/10.1109/ANDESCON.2010.5630082

17. Michael A. Goodrich and Alan C. Schultz. 2007. Human-Robot Interaction: A Survey. *Foundations and Trends® in Human-Computer Interaction* 1, 3 (2007), 203–275. DOI:http://dx.doi.org/10.1561/1100000005

18. Kel Guerin. 2017. Openni2-Tracker: A ROS Wrapper for the OpenNI2 and NiTE2 Skeleton Tracker. (2017). https://github.com/futureneer/openni2-tracker

19. Hamburg News. 2016. Starship Robot Delivers Hermes Packages in Hamburg. *Hamburg News* (2016). http://www.hamburg-news.hamburg/en/cluster/media-it/starship-robot-delivers-hermes-packages-hamburg/

20. Hideo Hattori. 2017. Autopep8. (2017). https://pypi.python.org/pypi/autopep8

21. Héctor Herrero, Jose Luis Outón, Urko Esnaola, Damien Sallé, and Karmele López de Ipiña. 2015. State Machine Based Architecture to Increase Flexibility of Dual-Arm Robot Programming. In *Bioinspired Computation in Artificial Systems (LNCS)*, Vol. 9108. Springer, Cham, Elche, Spain, 98–106. DOI: http://dx.doi.org/10.1007/978-3-319-18833-1_11

22. Seth Lloyd. 2001. Measures of Complexity: A Nonexhaustive List. *IEEE Control Systems Magazine* 21, 4 (2001), 7–8. DOI: http://dx.doi.org/10.1109/MCS.2001.939938

23. Alexandru Mandes and Peter Winker. 2016. Complexity and Model Comparison in Agent Based Modeling of Financial Markets. *Journal of Economic Interaction and Coordination* (2016), 1–38. DOI: http://dx.doi.org/10.1007/s11403-016-0173-0

24. Maja J. Matarić and François Michaud. 2008. Behavior-Based Systems. In *Springer Handbook of Robotics*. Vol. Part E, 38. Springer Berlin Heidelberg, 891–909. DOI: http://dx.doi.org/10.1007/978-3-540-30301-5_39

25. Pedro Tomás Mendes Resende. 2014. *Reinforcement Learning of Task Plans for Real Robot Systems*. MSc Thesis. Técnico Lisboa, Lisbon, Portugal.

26. Piaggio Fast Forward. 2017. Official Website of Gita. (2017). http://piaggiofastforward.com/gita

27. Werner Pluta. 2017. Piaggio: Roboter Gita Fährt Die Einkäufe Nach Hause. *Golem.de* (2017).

28. David Pogue. 2016. Exclusive: Amazon Reveals Details About Its Crazy Drone Delivery Program. *Yahoo!Tech* (2016). https://www.yahoo.com/tech/exclusive-amazon-reveals-details-about-1343951725436982.html

29. PyCQA. 2017. Pylint - Code Analysis for Python. (2017). https://www.pylint.org/ https://github.com/PyCQA/pylint.

30. Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. 2009. ROS: An Open-Source Robot Operating System. In *ICRA Workshop on Open Source Software*, Vol. 3. IEEE, Kobe, Japan, 6.

31. Anand S. Rao. 1996. AgentSpeak(L): BDI Agents Speak Out in a Logical Computable Language. In *Agents Breaking Away (LNCS)*, Vol. 1038. Springer, Berlin, Heidelberg, Eindhoven, The Netherlands, 42–55. DOI: http://dx.doi.org/10.1007/BFb0031845

32. Corrado Santoro. 2016. PROFETA - Python RObotic FramEwork for wriTing strAtegies. (2016). https://github.com/corradosantoro/profeta

33. Heiner Schmidt. 2016. Hermes startet in Hamburg Paketzustellung per Roboter. *Hamburger Abendblatt* (2016). https://www.abendblatt.de/wirtschaft/article207997697/Hermes-startet-in-Hamburg-Paketzustellung-per-Roboter.html

34. Hedvig Sidenbladh, Danica Kragic, and Henrik I. Christensen. 1999. A Person Following Behaviour for a Mobile Robot. In *IEEE International Conference on Robotics and Automation (ICRA)*, Vol. 1. IEEE, Detroit, MI, USA, 670–675. DOI: http://dx.doi.org/10.1109/ROBOT.1999.770052

35. Starship Technologies. 2017. Official Website of Starship Technologies. (2017). https://www.starship.xyz/

36. Sebastian Stock, Martin Guenther, and Joachim Hertzberg. 2014. Generating and Executing Hierarchical Mobile Manipulation Plans. In *International Symposium on Robotics (ISR) and German Conference on Robotics (ROBOTIK)*, Vol. 41. VDE VERLAG GMBH, Munich, Germany, 605–610. http://ieeexplore.ieee.org/document/6840188/

37. Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. 2001. Robust Monte Carlo Localization for Mobile Robots. *Artificial Intelligence* 128, 1 (2001), 99–141. DOI: http://dx.doi.org/10.1016/S0004-3702(01)00069-8

38. Johannes Twiefel, Timo Baumann, Stefan Heinrich, and Stefan Wermter. 2014. Improving Domain-Independent Cloud-Based Speech Recognition with Domain-Dependent Phonetic Post-Processing. In *AAAI Conference on Artificial Intelligence*, Vol. Twenty-Eighth. AAAI Press, Québec City, Québec, Canada, 1529–1535.

39. Guido van Rossum, Barry Warsaw, and Nick Coghlan. 2001. PEP 8 – Style Guide for Python Code. (2001). https://www.python.org/dev/peps/pep-0008/

40. Paul Viola and Michael Jones. 2001. Rapid Object Detection Using a Boosted Cascade of Simple Features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Vol. 1. IEEE, Kauai, Hawaii, USA, 511–518. DOI: http://dx.doi.org/10.1109/CVPR.2001.990517

41. Iris Wieser, Sibel Toprak, Andreas Grenzing, Tobias Hinz, Sayantan Auddy, Ethem Can Karaoğuz, Abhilash Chandran, Melanie Remmels, Ahmed El Shinawi, Josip Josifovski, Leena Chennuru Vankadara, Faiz Ul Wahab, Alireza M. Bahnemiri, Debasish Sahu, Stefan Heinrich, Nicolás Navarro-Guerrero, Erik Strahl, Johannes Twiefel, and Stefan Wermter. 2016. A Robotic Home Assistant with Memory Aid Functionality. In *KI 2016: Advances in Artificial Intelligence (LNCS)*, Vol. 9904. Springer International Publishing, Klagenfurt, Austria, 102–115. DOI:http://dx.doi.org/10.1007/978-3-319-46073-4_8

42. Mihalis Yannakakis. 2000. Hierarchical State Machines. In *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics (Lecture Notes in Computer Science)*, Vol. 1872. Springer, Berlin, Heidelberg, Sendai, Japan, 315–330. DOI: http://dx.doi.org/10.1007/3-540-44929-9_24