Aspects of Integrating Diverse Software into Robotic Systems
Extended Abstract

Tim Niemueller
Knowledge-based Systems Group
RWTH Aachen University
Aachen, Germany
niemueller@cs.rwth-aachen.de

Gerhard Lakemeyer
Knowledge-based Systems Group
RWTH Aachen University
Aachen, Germany
gerhard@cs.rwth-aachen.de

Alexander Ferrein
Electrical Engineering Department
Aachen University of Applied Sciences
Aachen, Germany
ferrein@fh-aachen.de

A robot platform consists of numerous devices such as sensors to acquire data or actuators to interact with the outside world. As such, a user of a new robot platform (that she most often has not built by herself) needs a software system with many drivers to control at least the lowest hardware layer of the system and a lot of software implementing algorithms for tasks like perception or locomotion.

The vendor of a robot platform has to come up with good selling points for his product which, at least in general and nowadays, is not a mass product but has a rather high price. Such systems should be open to extensions made by the customer for future developments and solving new tasks. One selling point here is to provide a complete software stack running on the platform so that the robot can be used out-of-the-box at least for certain tasks.

It is especially this desire for a ready-to-run system that often makes it cumbersome to adapt and extend the robot. During software integration the engineers of the system had certain scenarios in mind and assumptions were made required to avoid an open-ended development. This means that the user of the platform has to adapt to the platform first, before being able to start the own actual work.

Two examples for such platforms are the Nao[1] and the Robotino[2]. The humanoid robot Nao from Aldebaran is used in the RoboCup Standard Platform League and employs NaoQi [1]. It is a service-oriented framework for which modules must be written which in turn can then interact with the hardware and software components like walking motion pattern generators. The holonomic omni-drive robot Robotino from Festo Didactic is mostly used for educational purposes and in the new RoboCup Logistics League Sponsored by Festo [2]. It uses OpenRobotino [3], a messaging-based framework where clients connect to a central device server to access the hardware. Both platforms expect developers to use their existing framework as the only way to access the underlying hardware. The advantage of these systems is that newcomers can get started quickly. On the downside to use another framework (for example to re-use existing software), time must be spent to integrate the two systems. It becomes particular difficult if these framework are not Open Source software like NaoQi, for instance. NaoQi makes it particularly hard because it cannot be embedded in other software and remote network access is rather inefficient. Another software framework is the ROS framework [4]. While it has many great features and ready-to-use packages for typical robotic problems, one needs to be conformant with their middleware. Robots like the YouBot[3] come prepared to run with only this framework. Using any other requires a tremendous amount of effort. At least ROS provides efficient remote access and thin client libraries, making integration considerably easier like, say, NaoQi.

On the other hand, hardware components like sensors or actuators usually come with a software library to access data and give commands. Examples are the URG library[4] for Hokuyo laser range finders, or libdc1394[5] to access IEEE 1394 cameras. For such devices integration into a larger system is expected, therefore a thin interface is provided. Sometimes components for a particular framework are provided on top of such libraries[6], and sometimes such libraries are ignored a re-implemented to suit a particular framework.[7]

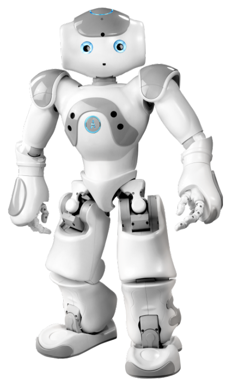
It is important to note that libraries and frameworks in general have an inversed control flow. Libraries provide classes or functions which are invoked by some application, while frameworks most often take control and trigger modules by means of callbacks, notifications, or cyclic calls. Especially this control flow is what makes it often more difficult to integrate components of separate frameworks.

On our robots we primarily employ Fawkes [5], in particular because it was developed with knowledge-based systems for behavior control in mind and therefore fits our area of primary research. Because we employ Fawkes on a variety of robots, including the mentioned Nao and Robotino, and our custom-built domestic service robot Caesar [6] (cf. Figure 1), we needed to integrate Fawkes with all of the mentioned frameworks and libraries.

Some of the issues we experienced have been outlined

[1]http://www.aldebaran-robotics.com/en/
[2]http://www.robotino.de
[3]http://youbot-store.com/
[4]http://www.hokuyo-aut.jp/02sensor/07scanner/download/urg_programs_en/
[5]http://libdc1394.sourceforge.net
[6]http://trac.fawkesrobotics.org/wiki/Plugins/laser
[7]http://www.ros.org/wiki/hokuyo_node

(a) Domestic service robot Caesar


(b) Humanoid robot Nao

(c) A modified Robotino robot

Fig. 1. Robots running Fawkes contributing to the experience made when integrating with other software

in [7], for example fragmentation, duplication of efforts, and software lock-in. Following the idea of component-based software design [8], [9], [10] and considering the differences of libraries and frameworks, new robot platforms should provide all basic drivers in form of software libraries and then use these libraries for integration into a particular framework. In this way, the functionality could be more easily integrated into the system of choice. The user would not have to extract some driver functionality hidden deep in some framework function. Extracting the code is moreover only possible if one is lucky and the framework's sources are provided. If not, one is stuck with the third-party framework and all the problems that come with it.

This problem is observed also by several robot software component developers. For example in ROS, several components which were integrated into the system have been separated from the framework in current software versions. The Point Cloud Library [11] for instance was deeply integrated with ROS' communication framework. Now, it is available as a library which can be used with or without ROS. Similarly, OctoMap [12] was bundled with ROS previously. Today, it has the mentioned thin interface and has then been re-integrated with ROS on top of this.

In summary, to integrate diverse software into a robotic system, the drivers and components should provide thin interfaces which are not specific to one particular robot framework. Encapsulating the functionality in a software library leaves all options to the user. No particular assumptions about the communication infrastructure of some robot framework are made. This gives the user of the library the chance to integrate the functionality more easily into the target framework of choice.

References

[1] "NaoQi," http://www.aldebaran-robotics.com/en/Discover-NAO/Key-Features/NAOqi.html, last visited on 2013/03/15.

[2] T. Niemueller, G. Lakemeyer, and A. Ferrein, "Incremental Task-level Reasoning in a Competitive Factory Automation Scenario," in *Proc. of AAAI Spring Symposium 2013 - Designing Intelligent Robots: Reintegrating AI*, 2013.

[3] "OpenRobotino," http://www.openrobotino.org/, last visited on 2013/03/15.

[4] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[5] T. Niemueller, A. Ferrein, D. Beck, and G. Lakemeyer, "Design Principles of the Component-Based Robot Software Framework Fawkes," in *International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, 2010.

[6] A. Ferrein, T. Niemueller, S. Schiffer, and G. Lakemeyer, "Lessons Learnt from Developing the Embodied AI Platform Caesar for Domestic Service Robotics," in *Proc. of AAAI Spring Symposium 2013 - Designing Intelligent Robots: Reintegrating AI*, 2013.

[7] A. Makarenko, A. Brooks, and T. Kaupp, "On the Benefits of Making Robotic Software Frameworks Thin," in *International Conference on Intelligent Robots and Systems*, 2007, workshop for Measures and Procedures for the Evaluation of Robot Architectures and Middleware.

[8] M. D. McIlroy, "'Mass Produced' Software Components," *Software Engineering: Report On a Conference Sponsored by the NATO Science Committee*, pp. 138–155, 1968.

[9] D. Brugali and P. Scandurra, "Component-based robotic engineering (Part I)," *IEEE Robotics Automation Magazine*, vol. 16, no. 4, pp. 84–96, 2009.

[10] D. Brugali and A. Shakhimardanov, "Component-Based Robotic Engineering (Part II)," *IEEE Robotics Automation Magazine*, vol. 17, no. 1, pp. 100–112, 2012.

[11] R. B. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Proc. of the 2011 IEEE International Conference on Robotics and Automation (ICRA-2011)*, 2011.

[12] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees," *Autonomous Robots*, 2013.