Applications of Visualization Technology in Robotics Software Development

Max Reichardt, Lisa Wilhelm, Martin Proetzsch, Karsten Berns

Robotics Research Lab
Department of Computer Sciences
University of Kaiserslautern, Germany
{reichardt, lisa.wilhelm, proetzsch, berns}@cs.uni-kl.de

Abstract: Control software for experimental autonomous robots is typically complex and subject to frequent changes – posing numerous challenges for software engineers. When based on general-purpose robotics frameworks, significant parts of such systems are modular and data-flow-oriented – a natural fit for visualization in graph structures. In this paper, we present approaches to visualize different aspects of robotics software which proved helpful or even essential in our development process. Furthermore, we briefly introduce central abstractions in our software framework which greatly facilitate generic solutions.

1 Introduction

Developing control software for autonomous robots is a complex task. Many non-functional requirements such as efficiency, safety or fault-tolerance need to be dealt with. Therefore, many robot controls are based on robotic frameworks. These integrate commonalities such as network interfaces, provide various general-purpose components, and include tools that support application developers – possibly by visualizing certain aspects of a system.

Robotic applications are data-flow-oriented up to a certain degree. Sensors continuously provide values, while actuators regularly receive updated control values. This is reflected in frameworks which can roughly be divided into two groups regarding the interfaces of their basic application building blocks ("components", "modules", or "services"): object-oriented interfaces with method calls and connector-style interfaces with connectible pins. In the latter, whole applications are integrated by arranging entities in a data flow graph. This is especially suitable for visualization in graphs. Some frameworks even provide graphical programming facilities – not dissimilar to Simulink[1] or LabView[2]. This is a controversial topic in the robotics community – "one feature that is repeatedly proposed to make robot programming easier is graphical programming, i.e. building systems by connecting boxes with lines using some spatial layout tool. (...) As anyone who has

[1]http://www.mathworks.com/products/simulink/
[2]http://www.ni.com/labview

used Simulink knows, complex programs quickly lead to cluttered screens and so lots of time is spent arranging objects spatially: an arrangement that has no meaning at all for the code" [VG07].

The next chapter deals with software visualization in popular frameworks. Then, relevant abstractions in MCA, which we use for development, are introduced briefly – followed by a coverage of some of our recent projects which have proven helpful in developing and improving our software. Graphically rich tools for simulation or sensor data visualization are not covered, since they are not directly related to software engineering problems. Nevertheless, they are essential for the development process as well – particularly with respect to testing.

2 Visualization in Popular Robotic Frameworks

Numerous frameworks have been developed in the past. In fact, "It is only a small overstatement to say that almost every lab has brewed its own solution for robot control architecture, middleware and software integration concepts" [SP07]. This overview concentrates a few well-known frameworks. While most frameworks provide facilities for data visualization, robot operation, and simulators – visualization support directly targeting software engineering problems is less common. There are, however, many worthwile possibilities – some of which are discussed in the next chapters.

A major target of Microsoft's *Robotics Developer Studio* was making development of robotic applications simpler. Part of this effort is the Microsoft Visual Programming Language (VPL). It allows creating robotic applications graphically by instantiating building blocks and connecting their inputs and outputs. Explicitly targeting "non-programmers", however, this language is hardly used in professional robotic projects to our knowledge.

The *Player Project* [GVS$^+$01] is arguably the most well-known open source robotics toolkit. Minimalism and simplicity being major design goals, it provides interfaces to a wide range of robotics hardware without making prescriptions for the robotic applications actually using them. Naturally, it cannot provide tools for visualizing these applications.

The open source robotics framework *Orca* [BKM$^+$07] is an attempt to bring Component-Based Software Engineering to the robotics domain. It includes some tools for management of components, interaction with robots, and visualization of components' internal state. *OrcaView2D* is the most interesting regarding the latter – allowing any component to render information to a two-dimensional map view using a generic interface.

FlowDesigner[3] is a visual open source programming tool with similarities to Simulink and LabView. The *RobotFlow*[4] plugin is an attempt to utilize it for robotics software. It is closely related to the MARIE framework [CLR07] – a major attempt to integrate components from different frameworks. However, the latest release is from 2005.

[3]http://apps.sourceforge.net/mediawiki/flowdesigner/
[4]http://robotflow.sourceforge.net/

3 Visualizing MCA Applications

At our lab, we use the MCA2 framework that was developed at the FZI (Forschungszentrum Informatik) in Karlsruhe. Modifications lead to an independent branch (MCA2-KL[5]). With many ideas for improvements, a new major release is currently under development. Some of these improvements are already considered here.

Basic building blocks of every MCA application are *modules*. Such modules have input and output *ports* that consume or provide data[6]. Connections between modules are network-transparent, so distributed applications can be easily created. To structure applications, modules may be grouped in *groups*. Basically, a set of connected modules form an application. On this level, an MCA application can be seen and visualized as a data flow graph. With interfaces based on facilities provided by the framework, various kinds of data can be collected and visualized in such a graph.

MCA Browser: Visualizing an application as a data flow graph is implemented in the `MCABrowser` tool. It is frequently used in our development process and particularly useful for checking that all modules are instantiated and connected correctly – as well as tracking down malfunctioning modules that publish incorrect values. The tool is occasionally improved and extended.

iB2C Extension: A recent such extension is the *iB2C* mode (see Fig. 1) for visualizing behavior-based architectures [PLB07]. It displays additional information for modules that implement a behaviour. This includes a behaviour's *activity*, indicating its influence on the system's control output, and the *target rating* representing its contentment with the current situation. The *activation* is determined by incoming signals and is an upper bound for the activity. These values – which are common for all behaviour modules – are represented as colored horizontal lines as illustrated in Fig. 1, providing a quick overview of each behaviour's state.

System Partitioning: A recurring question in our development process is how to split up distributed applications. Network connections introduce latency and have limited capacity. Modules that exchange a lot of data should preferably execute on the same system. The same is true for modules that are part of a tight control loop – especially if there are real-time requirements. Deciding which modules to run on which systems tends to become less straightforward as systems grow. The long-term goal is that the framework is able to do this automatically. Meanwhile, there is a recent experimental enhancement to the `MCABrowser` that is meant to assist a developer with respect to this topic. It visualizes the traffic between different modules (see Fig. 1). The traffic has two dimensions – frequency of data exchange and size of transferred data per second. The connection lines in the `MCABrowser` are colored blue when data is sent scarcely and red when exchanged frequently. The average data size is reflected in the lines' thickness.

Generic Graphical User Interfaces: Graphical user interfaces are regularly required when testing and deploying a robot. Creating and changing these can be tedious – espe-

[5]http://rrlib.cs.uni-kl.de/

[6]In MCA2, such port data was limited to numerical values, whereas the next release supports objects of arbitrary types. To exchange more complex data before, *blackboards* – areas of shared memory – had to be used.

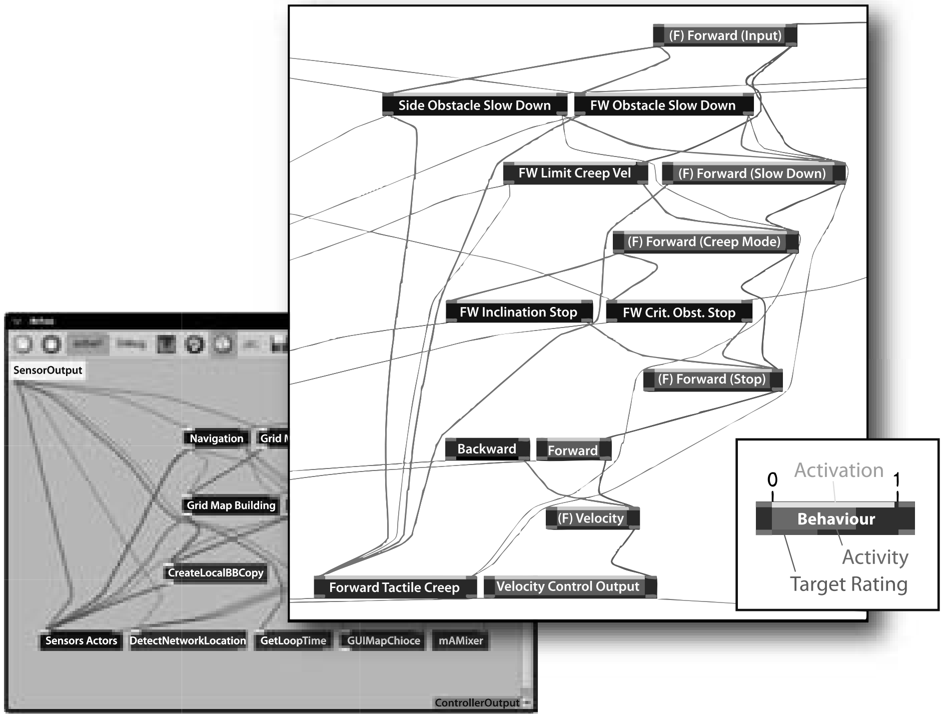

Figure 1: `MCABrowser` extensions: traffic analysis (left), iB2C behaviour visualization (right)

cially, for instance, if they run inside a web browser or need to be very robust. Therefore, MCA2 contains a generic GUI editor – `MCAGUI`. Recently, a similar improved Java-based variant (`JMCAGUI`) was developed [KRB08]. Basically, these editors allow placing GUI elements (*widgets*) on a scalable canvas and connecting them to a robot's inputs and outputs. Using a plugin mechanism, further widgets can be added. The Java version also allows adding further communication wrappers – enabling to build GUIs for completely different systems. GUIs may be deployed using a web server – possibly running on the robot itself. In the new version of the framework, many internals such as loop times, profiling data, or network parameters are also published via ports and can be accessed from browser and GUIs. This allows to quickly visualize such data whenever needed – advancing the GUI editors to somewhat generic software visualization tools. Notably, the same port data can be visualized in various ways by using different widgets.

Experimental Object Visualization in Debuggers: To make the Java GUI editor as universal as possible, several interfaces were introduced – most importantly *Paintable* that many of our classes implement. Any and possibly multiple such objects can be rendered, moved, rotated and zoomed in a *Geometry Renderer* widget. However, this is not limited to GUI editors. Using a small plugin, such visualization also works in the Eclipse debugger. This can aid developers in the debugging process with more intuitive data representations and can be seen as an interesting addition to an object's `toString`-method.

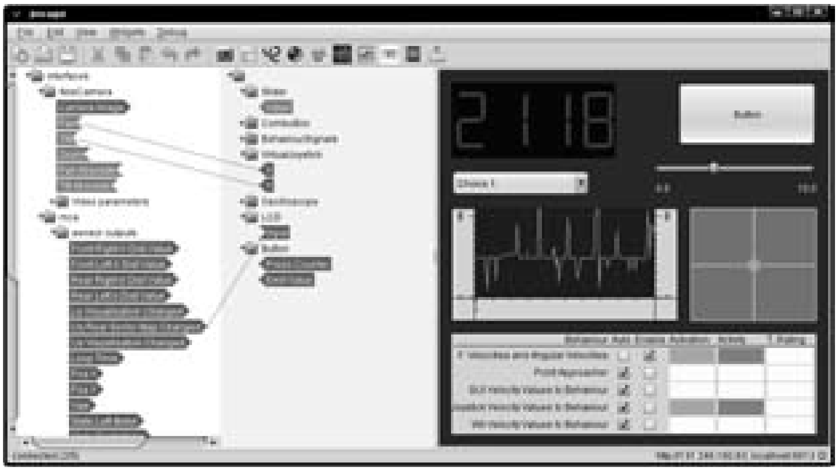

Figure 2: `JMCAGUI` GUI editor

Experimental Visual Programming: Modules are connected by adding appropriate statements to an application's source code. This can become tedious. Therefore, a graphical tool was implemented and empirically evualated in [Kna07] showing promising results. Creating a new application, developers were both quicker and made less mistakes. However, round-trip engineering is hardly possible with MCA2, since there is no standardized way to add edges. This issue is tackled in the new framework release.

3.1 Oscillation Analysis

Robustness, performance and reliability are important properties of software systems. Unwanted oscillations, e.g. in the signal data or in the visible robot behaviour, disturb the system's activity and can cause massive performance problems. These problems are well-known in mechanical and electrical engineering. There, a lot of research is done to optimize processes leading to many techniques for detecting and diagnosing oscillations [OT06, KCD$^+$04]. For software systems, oscillation detection is rarely discussed, although it can be very interesting. The algorithm outlined here was developed on the basis of techniques used by electrical engineers and adapted to the behavior-based robot control system iB2C [Wil08].

Oscillation Detection Procedure: Based on [THZ03], Fast Fourier Transform (FFT) is used to map the signal to the frequency domain and to detect peaks in the normalized power spectrum. To classify detected oscillations, two properties – *regularity* and *power* – are introduced. Both are based on a filtered spectrum which contains only a certain interval around the related peak. The *power* is calculated by summing up the spectrum's amplitudes. An oscillation's *regularity* is identified by transforming the spectrum back to time domain and analyzing zero-crossings in the signal. Regular and powerful oscillations

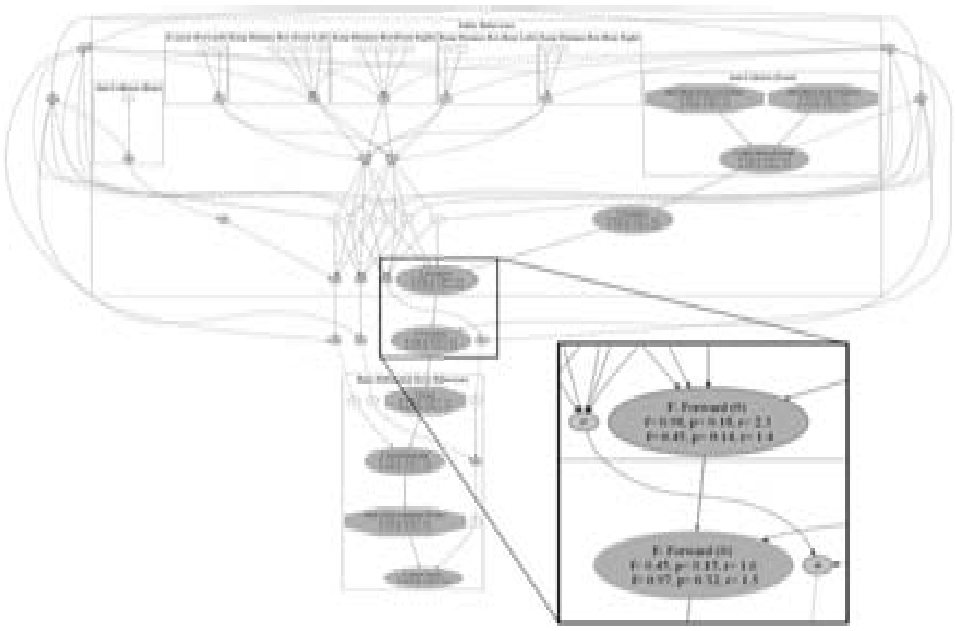

Figure 3: Control network of the mobile robot ARTOS visualized with Graphviz

are typically more important than weak, non-regular oscillations and should be analyzed further.

Visualization: The Boost Graph Library[7] (BGL) is used for the internal representation of the network. Fig. 3 shows a visualized control network that was created using the graph visualization software (Graphviz)[8]. Groups are illustrated as black boxes. Modules are represented as nodes labeled with their name and possibly frequencies, powers and regularities of any detected oscillations. Oscillating modules are coloured purple. The figure shows the path of an oscillation that is initially detected in a module in the middle of the network (red border). It can then be traced upwards to another group – the oscillation source – as well as downwards to a group responsible for controlling actuators.

4 Conclusion and Outlook

We presented several visualization approaches that help us coping with the complexity of robotic applications. They scale reasonably well with growing systems. We claim that a suitable uniform architecture, framework, or middleware is a fundamental factor for these kinds of software visualization tools. It allows collecting various types of data in a uniform way and across multiple projects – the latter increasing the return of investment for

[7]www.boost.org
[8]www.graphviz.org

developing such tools. There are certainly many more interesting areas for visualization such as, for instance, profiling – e.g. illustrating how much processing power modules consume and how much latency they introduce. In a current project, we plan improving our tools in cooperation with our university's computer graphics group. Early ideas include overcoming group boundaries in the `MCABrowser` tool, as well as making it simpler to add further visualization extensions.

References

[BKM+07] A. Brooks, T. Kaupp, A. Makarenko, S. Williams, and A. Orebäck. Orca: A Component Model and Repository. In Brugali [Bru07].

[Bru07] D. Brugali, editor. *Software Engineering for Experimental Robotics*, volume 30 of *Springer Tracts in Advanced Robotics*. Springer - Verlag, Berlin / Heidelberg, April 2007.

[CLR07] C. Coté, D. Letourneau, and C. Ra. Using MARIE for Mobile Robot Component Development and Integration. In Brugali [Bru07].

[GVS+01] B. Gerkey, R. Vaughan, K. Sty, A. Howard, G. Sukhatme, and M. Mataric. Most valuable player: A robot device server for distributed control. In *Proc. of the IEEE/RSJ Internatinal Conference on Intelligent Robots and Systems (IROS)*, pages 1226–1231, Wailea, Hawaii, October 2001.

[KCD+04] V. Kariwala, M. Choudhury, H. Douke, S. Shah, H. Takada, J. Forbes, and E. Meadows. Detection and Diagnosis of Plant-Wide Oscillations: An Application Study, 2004.

[Kna07] C. Knapwost. Forward und Reverse Engineering in der Robotik. Diploma thesis, Robotics Research Lab - University of Kaiserslautern, Mai 20 2007. unpublished.

[KRB08] J. Koch, M. Reichardt, and K. Berns. Universal Web Interfaces for Robot Control Frameworks. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nice, France, September 22-26 2008.

[OT06] P.F. Odgaard and K. Trangbaek. Comparison of Methods for Oscillation Detection - Case Study on a Coal-Fired Power Plant. In *Proceedings of IFAC Symposium on Power Plants and Power Systems Control 2006*, 2006.

[PLB07] M. Proetzsch, T. Luksch, and K. Berns. The Behaviour-Based Control Architecture iB2C for Complex Robotic Systems. In *30th Annual German Conference on Artificial Intelligence (KI)*, pages 494–497, Osnabrück, Germany, September 10-13 2007.

[SP07] A. Shakhimardanov and E. Prassler. Comparative Evaluation of Robotic Software Integration Systems: A Case Study. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007)*, San Diego, CA, USA, October 29-November 2 2007.

[THZ03] N.F. Thornhill, B. Huang, and H. Zhang. Detection of Multiple Oscillations in Control Loops. *Journal of Process Control*, 13:91–100, 2003.

[VG07] R. Vaughan and B. Gerkey. Reusable Robot Software and the Player/Stage Project. In Brugali [Bru07].

[Wil08] L. Wilhelm. Oscillation Detection in Behaviour-Based Robot Architectures. Diploma thesis, University of Kaiserslautern, November 2008. unpublished.