On B-Human's Code Releases in the Standard Platform League – Software Architecture and Impact

Thomas Röfer and Tim Laue

Deutsches Forschungszentrum für Künstliche Intelligenz,
Cyber-Physical Systems, Enrique-Schmidt-Str. 5, 28359 Bremen, Germany
{Thomas.Roefer, Tim.Laue}@dfki.de

Abstract. In RoboCup, source code releases after a competition are one important part of the competition's overall progress. In particular teams in the Standard Platform League can strongly benefit from other's software, as everybody shares the same robot platform. Therefore several releases by different teams exist. In this paper, we describe the past code releases of our team B-Human, particularly focusing on its underlying software architecture, which is set in relation to concepts of the currently popular Robot Operating System (ROS), as well as on its impact on the league's progress.

1 Motivation

B-Human started as a team in the Humanoid League. We adapted the GermanTeam framework for our Bioloid-based robots [10] before we switched to the Standard Platform League (SPL) in 2009 and thereby to the NAO. Our history in the Humanoid League gave us a certain advantage in the Standard Platform League, because our main weakness in the Humanoid League was the robot hardware, which – in contrast – is identical for all teams in the SPL. B-Human won all official games it played so far except for the final in RoboCup 2012. As a result, the team won the RoboCup German Open five times (2009–2013) and the RoboCup World Championship three times (2009, 2010, 2011). Although code releases are not mandatory in the SPL, B-Human has been releasing its software since the league started to use the NAO as standard platform in 2008 [11] at `www.b-human.de/publications`. We believe that providing software as Open Source is the best way to foster research and push a RoboCup league forward.

The remainder of this paper is as follows: Section 2 presents the technical aspects of the B-Human framework which are compared to the popular Robot Operating System in the following Section 3. The impact of the B-Human code releases is described in Section 4. Finally, the paper is concluded in Section 5.

2 B-Human's Software Framework

B-Human's software framework is based on the framework of the GermanTeam in the Four-Legged League [10]. The GermanTeam was a joint team of four (later

three) universities that competed separately at local events, but participated as a single team at the RoboCup. Some elements of the software framework are motivated by these circumstances, namely that several modules which provide very similar functionality can exist in parallel, because they were originally developed by members of different university teams for a local competition.

Since the GermanTeam framework always aimed at being as hardware-independent as possible, porting it from AIBO to Bioloid to NAO was not very difficult. Such transitions would have been a lot more complicated if we had directly used Sony's Open-R on the AIBO that does not exist anywhere else. Therefore, we also abstract from Aldebaran Robotics' NaoQi, because the league might switch to a different robot platform in the future.

Since 2011, the architecture that is now called *B-Human framework* supports the three most popular operating systems as development platforms. Microsoft Windows running Visual Studio, OS X running Xcode, and Linux are all treated as first class citizens in the development process. To reduce the logistic overhead that comes with this variety, we recently switched to llvm/clang as C++ compiler. It can cross-compile the code for the robots on all three platforms, it is Xcode's native compiler, and it works with any development environment on Linux. Only when targeting Windows, Microsoft's compiler is still used. In addition, we increasingly use the expressiveness of C++ and its preprocessor to add new features rather than employing external tools such as pre-compilers. Such tools complicate the build process and they always need to be available for all three platforms. In 2013, even the modeling of the robots' behavior using hierarchical state machines is done in C++, still following the same general approach that B-Human and the GermanTeam used since 2002 [7], but avoiding the overhead of coupling two different programming languages together.

2.1 Processes

A robot control program reacts to external events, i. e. when new sensor data was acquired or when the robot requires new target values for its actuators. In our framework, *processes* are employed to promptly react to these events. These framework processes can be either implemented by using operating system processes or by using threads. The former was the case on the GermanTeam's AIBOs and on our robots in the Humanoid League, the latter is the case in B-Human's current implementation on the NAO.[1] The NAO offers two kinds of sensor information: camera images from *Video for Linux* and everything else from NaoQi's *Device Control Manager*, i. e. actual joint angles, IMU and sonar measurements, etc. The latter is offered at the same speed as target joint angles are requested. Therefore, the framework uses two processes, namely *Cognition* that runs at the speed at which camera images are taken (30 Hz for NAO V3.x, 60 Hz for NAO V4) and *Motion* that runs at the speed the other sensor measurements are provided, i. e. 100 Hz. A third process *Debug* offers a network connection to an external PC and is used for debugging purposes only (cf. Fig. 1).

[1] To keep consistent between different instantiations of the framework, we still use the term *process*, although the current implementation is based on threads.

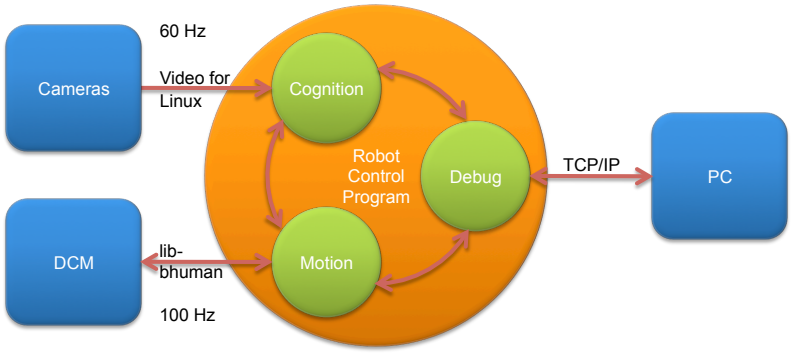

Fig. 1. The processes used on the NAO

2.2 Data Representation

B-Human's software architecture consists of *modules* that implement the functionality (cf. next section). These modules exchange data in the form of *representations*. Representations are instances of C++ classes that mainly store data and provide nearly no functionality. For instance, there are representations that store objects detected in an image, e. g. the ball, models of objects in the environment, e. g. the location of the ball on the field (Fig. 2 contains a simplistic example), or requests from one module to another one, e. g. to walk with a certain speed.

All representations are stored in a blackboard [5]. Each process has its own blackboard that contains all representations that are used in this process. The representations are either provided by modules that run in the same process or received from another process. Since each process has a copy of each representation that is processed by its modules, there is no concurrent access to the representations, which guarantees consistency, i. e. the contents of a representation provided in a different process will always be the same through a whole

```
class BallModel : public Streamable {
  void serialize(In* in, Out* out) {
    STREAM_REGISTER_BEGIN
    STREAM(position)
    STREAM(wasLastSeen)
    STREAM_REGISTER_FINISH
  }
public:
  Vector2<float> position;
  unsigned wasLastSeen;
};
```

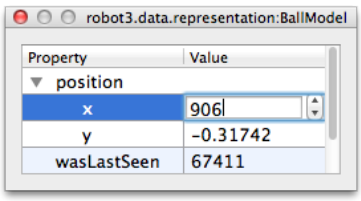

Fig. 2. Simple example of the definition of a representation on the left (only #include statements omitted) and its live visualization on the right

processing cycle. At the end of each processing cycle, representations that are required by the other process are simply sent to it, which it receives at the beginning of its next processing cycle.

All representations are streamable, i. e. they can be written to and read from a data stream. The C++ macro (cf. Fig. 2 left) used to stream attributes of representations does not only handle reading and writing, it also provides the name of the attribute and its data type at runtime. On the one hand, this is used to read and write structured, human-readable configuration files without any further code necessary. On the other hand, any representation can be visualized and edited at runtime through a debugging tool on a PC (cf. Fig. 2 right).

2.3 Components Performing Computations

In the B-Human framework, computations are performed by *modules*. Each module requires a number of representations as input to perform its job and it provides one or more representations as output to other modules. The dependencies between requirements and outputs define an execution sequence for all modules that is automatically determined, i. e. the execution of the modules is automatically ordered in a way that it is guaranteed that before a module is executed, all of its requirements have already been updated by other modules. Thereby, inconsistent configurations are detected and reported to the programmer. It is also automatically computed, which of the representations have to be exchanged between the different processes. Figure 3 shows an example of the dependencies between modules and representations.

The architecture allows that several modules exist that provide the same representation, but only one of them can be active at the same time. The set of

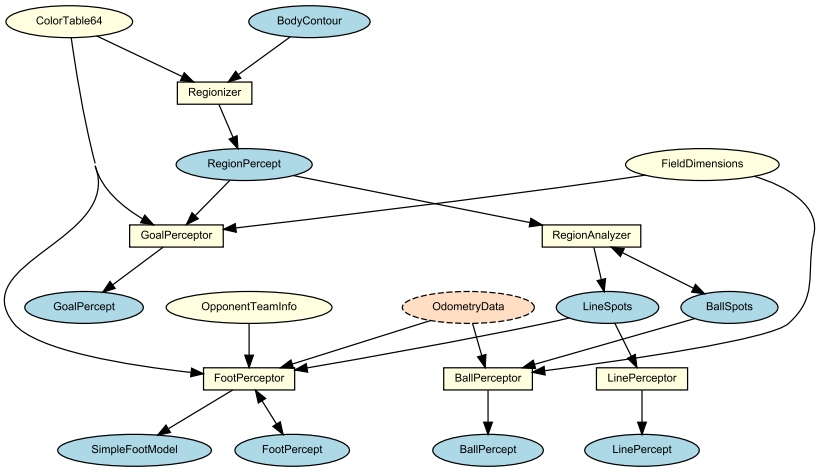

Fig. 3. A small subset of the modules (shown as rectangles) and representations (shown as ellipses – with a dashed border if received from another process) used. Dependencies on representations that are not depicted have been removed for a better readability.

active modules is specified in a configuration file. When a new module is developed, it can either be activated by changing the configuration file or interactively while the B-Human software is running, and thereby recomputing the execution sequence of the modules and rejecting inconsistent configurations. This eases software development, because different implementations can exist in parallel.

Since it is the intention of the architecture to simplify replacing existing implementations with new and hopefully better ones, it is an important rule that both representations and modules should be small and coherent. Small representations such as the position of the ball relative to the robot allow implementing a module that computes exactly this information, which can easily be re-implemented in a new module that tries to follow a different approach. In contrast, a representation that would consist of everything soccer-relevant that can be found in an image (ball, lines, goals, robots, etc.) would require for a single module to provide all this information, making it highly complex and hard to replace by a different approach. Therefore, B-Human's software consists of a large number of small representations and rather small modules. For instance, currently there are nearly 100 modules in the system that exchange approximately 120 representations.

3 Comparison to the Robot Operating System (ROS)

A robotics framework that gained much attention in recent years and has also been used on the NAO robot by some SPL teams is the Robot Operating System (ROS) [9]. Similarly to the B-Human framework, it provides a clear encapsulation of computing components and the data representation exchanged among these components. However, the underlying concepts and intentions – i. e. *nodes* and *messages* versus *modules* and *representations* – differ from each other.

In ROS, computations are carried out by the so-called *nodes* with each node being a single (operating system) process. The corresponding *modules* in B-Human's framework do not run in parallel to each other but are grouped in a fixed number (currently two) of processes. This is – of course – less flexible and less suitable given a scenario that involves a number of available CPU cores and associated enormous computations. However, the NAO has only a single CPU core (with Hyper-Threading in case of the V4) that is not even used to full capacity by our software. We do not expect this situation to change significantly in the near future. Our current module layout focuses on a very fine-grained modularization which allows to put even trivial functionalities (such as computing the overall walked distance based on odometry offsets provided by another module) into separate modules. This can be done without losing reactivity and efficiency due to scheduling and communication overheads which would occur in case of a direct mapping of our current modules to ROS nodes.

The ROS counterpart to B-Human's *representations* are *messages* (also referred to as *topics* after instantiation). Both concepts are quite similar and provide equal features such as streaming or the possibility of replacing senders and receivers during run-time. The main difference is the way of defining the data

structures. ROS messages are specified in external files, which are used to generate actual code. This allows ROS to support different programming languages. As aforementioned, we aim to keep the number of external tools, languages, and compilers as low as possible. That is why B-Human's representations are specified directly in C++. In theory, this allows to add computational functionality to the representations, but in general, we refuse to use this option, because in contrast to modules, the implementations of representations cannot be switched at runtime.

4 Impact

Since 2008, we released a documented software package after each year's RoboCup world championship. They found broad interest in the RoboCup community and many teams use the B-Human software or certain parts of it.

4.1 Release Procedure and Impact on Our Own Development

The B-Human software is released every fall, shortly before the winter term starts. Each release consists of two parts: a software archive and a team report that includes detailed installation and operating instructions. We do not maintain any public repository as some other teams do. Hence, we do not publish any bug fixes for our released code but so far also never received any bug fixes from teams that use our code.

A code release always contains all components that have been used in the previous competition (without any modifications beyond bug fixes) except for the behavior, some special kicking motions, or components that are part of a thesis in progress. We always release our code under a BSD-style license as this grants the users the most liberties. However, we added some special RoboCup clauses to the license: Each team that uses B-Human code has to a) announce this on the league's mailing list prior to the next competition it takes part in and b) replace more than one major part of the module stack. As the B-Human team has been very successful in the past years, we consider this combination of a slightly trimmed code release and a demanding license to be the best trade-off between fostering the league's progress on the one hand and preventing a domination of B-Human clones on the other hand.

In addition to contributing to the league's progress as described in the following section, these code releases also benefit our own progress. On the one hand, the team reports serve as final reports of the current student project. On the other hand, we use our team reports internally to brief the new team members who join us every year. Instead of referencing a collage of research papers, README files, and Wiki pages as starting points, the latest team report serves as *one* single source including all necessary information for installing, using and understanding the B-Human system. Even experienced team members refer to the team report as reference as the system has grown to a complexity that makes it hard for anyone to know "everything". Therefore, we can recommend all teams to provide code releases and corresponding team reports.

4.2 Impact on Other Teams

In recent years, several other SPL teams used B-Human's software. This has been done in two different ways: Using the whole B-Human system and replacing single parts or using an own framework and integrating specific B-Human components. From a software engineering point of view, the former is quite easy due to the highly modular framework which allows a straight forward addition and replacement of new modules and representations. The latter is comparatively difficult as most modules have – despite their algorithmic and semantic enclosure – many dependencies to B-Human-specific implementations for streaming, mathematical computations, and debugging.

At least seven teams have based their development on the B-Human framework, including *NTU Robot PAL* [12] (reaching the third place at RoboCup 2011), *BURST* [6], *MRL* [4], and *NimbRo* (the runner-up at the RoboCup German Open 2010 and 2011). The latter also based research on the framework and the output of certain B-Human modules [8].

At least another seven teams use parts of our code. The most popular one used without the framework appears to be the implementation of the walking gait that has been presented by Graf and Röfer [3]. Amongst others, the current SPL world champion *UT Austin Villa* [2] uses the B-Human 2011 gait, relying on a version provided by the *Northern Bites* team at `github.com/northern-bites`.

Applications and connections outside the SPL include the usage of the walking gait by the NUbots in the Humanoid League [1] as well as the integration of a client for the Small Size League's standard vision system SSL-Vision [13] in the B-Human software. This allows the usage of SSL-Vision as a source for ground truth data for experiments with NAO robots.

5 Conclusions and Future Works

B-Human's software framework is well suited for controlling soccer robots that are equipped with embedded computers of rather low performance. It supports the rather dynamic development necessary to successfully participate in a robot soccer competition with continuously evolving rules. Due to its yearly releases, the software has been used by quite a number of other RoboCup teams as well.

We recently considered the option to switch to ROS but its strengths and preferences do not fit the way we currently want to lay out our software. In addition, the ROS support for some of our development platforms is still experimental, whereas our framework supports Linux, Windows, and OS X in equal measure without preferring any of these. However, in the near future, we plan to integrate other Open Source software such as *Eigen*.

References

1. Annable, B., Budden, D., Calland, S., Chalup, S.K., Fenn, S.K., Flannery, M., Fountain, J., King, R.A., Mendes, A., Metcalfe, M., Nicklin, S.P., Turner, P.,

Walker, J.: The NUbots team description paper 2013. In: RoboCup 2013: Robot Soccer World Cup XVII Preproceedings. RoboCup Federation, Eindhoven, Netherlands (2013), to appear
2. Barrett, S., Genter, K., He, Y., Hester, T., Khandelwal, P., Menashe, J., Stone, P.: UT Austin Villa 2012: Standard platform league world champions. In: Chen, X., Stone, P., Sucar, L.E., der Zant, T.V. (eds.) RoboCup 2012: Robot Soccer World Cup XVI. Lecture Notes in Artificial Intelligence, Springer (2013), to appear
3. Graf, C., Röfer, T.: A center of mass observing 3D-LIPM gait for the RoboCup Standard Platform League humanoid. In: Röfer, T., Mayer, N.M., Savage, J., Saranli, U. (eds.) RoboCup 2011: Robot Soccer World Cup XV. Lecture Notes in Artificial Intelligence, vol. 7416, pp. 101–112. Springer
4. Hashemi, E., Ghiasvand, O.A., Jadidi, M.G., Karimi, A., Hashemifard, R., Lashgarian, M., Shafiei, M., Farahani, S.M., Zarei, K., Faraji, F., Harandi, M.A.Z., Mousavi, E.: Mrl team description 2010 standard platform league. In: Chown, E., Matsumoto, A., Ploeger, P., del Solar, J.R. (eds.) RoboCup 2010: Robot Soccer World Cup XIV Preproceedings. RoboCup Federation, Singapore (2010)
5. Jagannathan, V., Dodhiawala, R., Baum, L.S. (eds.): Blackboard Architectures and Applications. Academic Press, Boston (1989)
6. Keidar, M., Aharon, I., Barda, D., Kamara, O., Levy, A., Polosetski, E., Ramati, D., Shlomov, L., Shoshan, J., Yakir, A., Zilka, A., Kaminka, G.A., Kolberg, E.: Robocup 2010 standard platform league team burst description. In: Chown, E., Matsumoto, A., Ploeger, P., del Solar, J.R. (eds.) RoboCup 2010: Robot Soccer World Cup XIV Preproceedings. RoboCup Federation, Singapore (2010)
7. Lötzsch, M., Risler, M., Jüngel, M.: XABSL - A pragmatic approach to behavior engineering. In: Proceedings of IEEE/RSJ International Conference of Intelligent Robots and Systems (IROS). pp. 5124–5129. Beijing, China (2006)
8. Metzler, S., Nieuwenhuisen, M., Behnke, S.: Learning visual obstacle detection using color histogram features. In: Röfer, T., Mayer, N.M., Savage, J., Saranli, U. (eds.) RoboCup 2011: Robot Soccer World Cup XV. Lecture Notes in Artificial Intelligence, vol. 7416, pp. 149–161. Springer
9. Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: Proceedings of the Open-Source Software workshop of the International Conference on Robotics and Automation (ICRA). Kobe, Japan (2009)
10. Röfer, T., Brose, J., Göhring, D., Jüngel, M., Laue, T., Risler, M.: GermanTeam 2007. In: Visser, U., Ribeiro, F., Ohashi, T., Dellaert, F. (eds.) RoboCup 2007: Robot Soccer World Cup XI Preproceedings. RoboCup Federation (2007)
11. Röfer, T., Laue, T., Burchardt, A., Damrose, E., Gillmann, K., Graf, C., de Haas, T.J., Härtl, A., Rieskamp, A., Schreck, A., Worch, J.H.: B-Human team report and code release 2008 (2008), only available online: `http://www.b-human.de/file_download/16/bhuman08_coderelease.pdf`
12. Wang, C.C., Wang, S.C., Yen, H.C., Chang, C.H.: NTU robot PAL 2009 team report (2009), only available online: `http://www.csie.ntu.edu.tw/~bobwang/RoboCupSPL/NTU_Robot_PAL_09Report.pdf`
13. Zickler, S., Laue, T., Birbach, O., Wongphati, M., Veloso, M.: SSL-Vision: The shared vision system for the RoboCup Small Size League. In: Baltes, J., Lagoudakis, M.G., Naruse, T., Shiry, S. (eds.) RoboCup 2009: Robot Soccer World Cup XIII. Lecture Notes in Artificial Intelligence, vol. 5949, pp. 425–436. Springer (2010)