Integrated Research and Development Environment for Real-Time Distributed Embodied Intelligent Systems

Antonello Ceravola, Frank Joublin, Mark Dunn, Julian Eggert, Christian Goerick

2006

Preprint:

This is an accepted article published in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. The final authenticated version is available online at: https://doi.org/[DOI not available]

Integrated Research and Development Environment for Real-Time Distributed Embodied Intelligent Systems

Antonello Ceravola, Frank Joublin, Mark Dunn, Julian Eggert, Marcus Stein and Christian Goerick
Honda Research Institute Europe GmbH
Offenbach/Main, Germany
Antonello.Ceravola@honda-ri.de

***Abstract* - In the field of intelligent systems, research and design approaches vary from predefined architectures to self-organizing systems. Regardless of the architectural approach, such systems may grow in size and complexity to levels where the capacities of people are strongly challenged. Such systems are commonly researched, designed and developed following several methods and with the help of a variety of software tools. In this paper we want to describe our research and development environment. It is composed of a set of tools that support our research and enable us to develop large scale intelligent systems used in our robots and in our test platforms. The main parts of our research and development environment are: the component models BBCM (Brain Bytes Component Model) and BBDM (Brain Bytes Data Model), the Middleware RTBOS (Real-Time Brain Operating System), the Monitoring system CMBOS (Control-Monitor Brain Operating System) and the Design Environment DTBOS (Design Tool for Brain Operating System). We will compare our research and development environment with others available on the market or still in research phase and we will describe some of our experiments.**

Index Terms - Parallel, Multithread, Multiprocessing, Real-Time, Modularity.

I. Introduction

Researchers around the world are continuously building intelligent systems that speak, see, learn, navigate and operate under conditions where even humans can't. Very few groups are focusing on building these systems in an integrated way with methods and tools that simplify handling the typical complexity and enable reuse. Researchers validate and implement their ideas through the usage of software technologies, given that, computers are, implicitly or explicitly, accepted as a general platform (very few researchers are building their own computing architectures like analog or neural computers). For these reasons we decided to introduce a new abstraction layer that on one side supports the design and the creation of large-scale systems and on the other side it is based on standard computers and standard operating systems.

A set of tools and computers alone cannot solve all the issues that usually emerge in researching and creating intelligent systems. It is clearly necessary to research the architectures, the methods and the approaches towards the creation of such large-scale systems. Here the effort is on the principles that regulate and organize the research work towards a coherent and consistent design and implementation of the many parts that an intelligent system is composed of. It is not the scope of this paper to illustrate and discuss such principles (for an introduction see [1]), while instead we will give an overview of the overall process we use in our software system research and the tools that supports us in the creation of real-time distributed embodied intelligent systems.

A. *Related Approaches*

In the recent years, in the field of intelligent systems, robotics and real-time systems, researchers started to build software environments like the one we describe in this paper. The approaches vary mainly in the abstraction level and the pursued targets. On a basic abstraction level, a modular approach is proposed by [2], [3] using a "component model" (software packaging patterns). We believe that this is a crucial aspect for intelligent systems since building system out of components/modules, allows cutting complex problems into smaller pieces, *reducing complexity*, increasing *reusability* and *decreasing dependency* (see [4], [5]). In [6] the usage of a set of libraries that supports data communication is proposed. Communication is one of the most important issues in modular systems, especially when building component-based systems, suppose to simulate brain-like functions. In [7] the usage of image/matrix manipulation libraries in combination with their development systems is proposed. The advantage here is that components/modules can benefit of a low level library that may gives a significant acceleration for common image/sound manipulation or matrix operations. Approaches, which are closer to the one we propose in this paper, are described in [8], [9], [10], [11], [12] where the aim is placed on an integrated environment which includes designing and monitoring tools. In these papers many different approaches are pursued, custom to specific problems or to particular architectures.

Generally, the most visible shortcoming of the cited solutions is that each of them is specialized for a particular domain or for a specific purpose. Moreover, some of the solutions described, put too much focus on specific features, overloading them with too much functionality, which, in practice, are not really necessary. We believe that it is crucial to have one single research and development environment that, while giving freedom to design and implement any

architecture (see [9] for similar arguments), provides all the necessary support for system decomposition, communication patterns, scalability into computers up to the support of execution with different sequential or parallel paradigms. Moreover, simplicity and modularity are, in this respect, a key point for the development and the usage of such research and development environments.

B. *Our Approach*

In Software Engineering it is well known that the creation of large-scale systems must be supported by a well defined process. Such process defines the various phases a project is composed of, the relation between such phases, the interaction and the tools required to accomplish the necessary tasks. We decided to employ such a process for our research in order to take advantage from it and structure our work. Our overall research and development process it's represented in Figure 1.

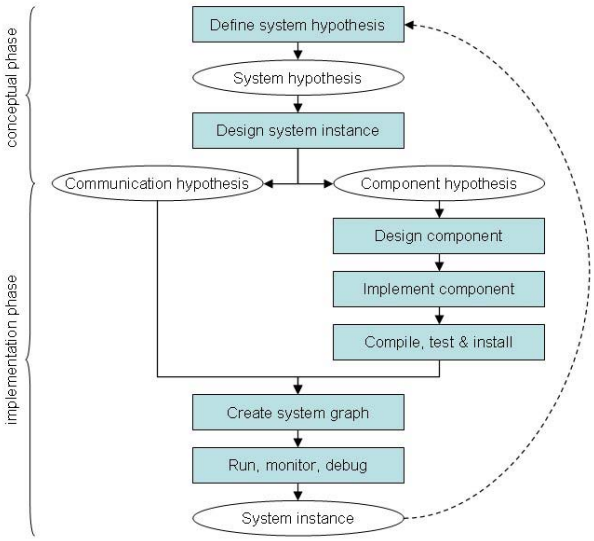

Figure 1: Our research and development process.

This process governs the life cycle of our systems research. An iteration of the full process runs on the time scale of about one year. Each step of this process is executed by several actors and is actually governed by sub-processes that determine the more fine grain actions. We distribute a set of check points in the process to ensure a certain level of quality at each step. The process starts with a conceptual phase where we maturate and formalize the ideas we want to realise in our systems. In this phase every researcher may use a different approach to reach his results. But in the transition phase (arrows in the graph) we coordinate and collect hypothesis into one common form. This is very important in order to cross-check the weak points or the unclear ones, or simply to confirm what seems to be accepted by the researchers. Other steps in the implementation phase are more concrete and require the creation of software modules, the testing of the functionality or the analysis of properties that modules should have. For such steps we decided to develop a set of tools that support the accomplishment of the related tasks. The major novel contributions of our system are: a) the partition we have created between components for algorithms (wrapping our algorithms, BBCM) and components for data (wrapping the communicated data between components, BBDM); b) the ways threads, execution patterns and processes are defined (controlled by RTBOS, Execution Patterns); c) the integration of all parts of our system into one single environment. The necessity of a) comes from the needs of controlling and configuring connections between components in a direct way. Additionally, components can ask for specific connection properties directly to the data component to which they are connected to. Moreover, through b) we do not require the creator of components (for data or algorithms) to code complex functions like input/output synchronization and /or parallelization handling. Such properties are defined in the application's graph and can be modified and reviewed any time without requiring a coding phase. The necessity of c) comes from the requirement of having all parts of the system working together with the minimum effort from the user point of view.

In section *II* we describe the steps of our process and the tools we have created for supporting them; in section *III* we review the history of the development of system hypothesis using this process and tools. Section *IV* summarizes and concludes the paper.

II. Our Process and Tools

Let's revisit the process of Figure 1 and analyse each step into more details:

A. *Define System Hypothesis*

This step belongs to the conceptual phase where the research work is concentrated on the definition of a system hypothesis. In this phase we elaborate the architecture, the functionality and the methods we want to research on. The goal here is to reach a coherent system hypothesis that contains all the functionalities and the principles we want to experiment on in an integrated concept.

B. *Design System Instance*

In this last step of the conceptual phase we develop a concrete design of the system. Here ideas and principles are mapped to concrete functions (component hypothesis) and relations (communication hypothesis). This step allows us to understand what type of components is required in order to develop the target system. Decomposing a system into components requires a certain balance between generality and specificity. The space of components can go from simple operation like addition, multiplication, to more complex one like filter banks, integrators, until components that perform object recognition or speech synthesis. The main driving forces to choose granularity are openness for extensions and performance. The systems we target are real-time systems working in real conditions in the real world. Decomposing a system in too fine grain components may overload the communication channels. This has to be balanced with the

choice of the employed communication paradigms. Communication between several components that use the memory channel of a single machine is much more convenient (in certain circumstances) then a communication channel over a network connection among computers. Here there is again a wide spectrum of possibilities. Network communication between computers can be implemented in many different ways, e.g. TCP, UDP and Shared Memory Cards.

In our experience we have had the needs to develop components which implements simple operations, and components that performs more complex functionality. The goal of this step is to identify the components we need and the communication patterns we want to employ for each part of the graph of the system. There we also identify which components can be reused and make a preliminary decomposition into computers. This preliminary partition is constrained by the actual availability of sensors and effectors.

C. *Design Component*

The implementation phase starts with the design of components. In our philosophy we clearly separate processing from communication, to allow for experimenting with different systems architectures, connectivity graphs and communication means without touching the internals of the processing algorithms. Therefore, we decided to encapsulate the processing (algorithms) into components. We have defined the BBCM (Brain Bytes Component Model) specification, a very simple but powerful component model that implements the interface shown in Figure 2a.

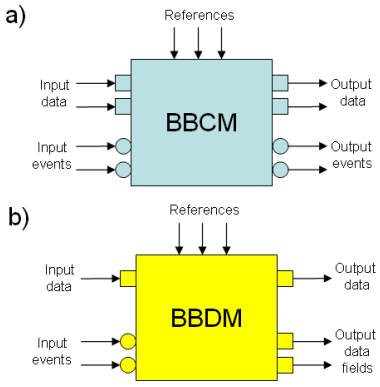

Figure 2: BBCM and BBDM interface.

All communication with a component is performed via this interface, which in the case of input and output are simple memory blocks handed over to the component from the outside. Events are data-less function calls. The BBCM has been implemented in the languages C, C++ and Matlab.

The data being communicated usually follow a data component specification for data types. It is called BBDM (Brain Bytes Data Model) which allow us to wrap any data type in standard components with a well defined interface (see Figure 2b).

The two component models (BBCM and BBDM) have been designed to work together (but they do not depend on each other). Furthermore a C BBCM and a C++ BBCM component can be used and can be connected together in the same source code file. The proposed standardization level is on the one side sufficiently strict in order to be a suitable basis for additional tools relieving much of the implementation burden from the researchers, and on the other side sufficiently flexible in order to allow for experimenting with different processing and communication paradigms.

D. *Implement Component*

This step consists in creating and coding the different components. It can be performed by the researcher or outsourced to external companies. This step is supported by different tools and conventions:

Coding and naming conventions: we decided to employ a set of conventions for our software. Through the coding conventions we improve readability and ensure a certain level of quality. Here we tried to keep the set of conventions as small as possible in order to minimize the effort to learn and use them. We support their usage by web tools.

Image and sound libraries: one of our main fields of research is image, sound and speech understanding and robot manipulation. For this reason we have created VLW (Visual Library Wrapper), and SLW (Sound Library Wrapper); libraries that support image and sound manipulation (e.g. arithmetic operations, filtering, transformation, conversions). These libraries are wrappers around existing libraries, providing a standard interface to our software. The wrapped libraries are: the IPP library [13], the Media Lib library [14] and a plain C version used in platforms that do not support IPP or Media Lib. These libraries are heavily used for building the algorithms wrapped in our BBCM components.

Template generator: since we often create components, we decided to use automatic code generators that generate the template for a BBCM or BBDM component from a simple textual description of its interface. We created SMDL (Simple Macro Descriptive Language), a language for transforming any document into a template. The usage of automatic code generators has many benefits, like:

- The generated code always conforms to our standards (conventions and libraries);
- The generated code is less prone to errors;
- Large amounts of source code can be generated in a fraction of time;

E. *Compile, Test and Install*

This traditional step is supported by several standard and custom tools that remove some of the load usually associated with this step in a multiplatform environment:

Versioning system: we are currently using SVN (Subversion [15]) for source code versioning. This tool (open-source, freely available on the internet) handles directory and source code file versioning in a local or networked configuration.

Multiplatform makefile system: we have based our build system on the GNU make [16]. Our build system is composed by two sets of files: a file *makefile* (read-only for users) which contains all rules we need for building, installing and handling the versioning system for all our modules; a file *makeVar* which contains user settings (include paths, libraries and compiler options). Our build system is multi-platform and supports until now projects for C, C++, PHP and Matlab CMEX.

Multiplatform deployment structure: in order to store and share our components, modules and applications we have created a multi-platform installation directory structure. This directory structure (that looks the same for all platforms we use) stores include files, object files, libraries, binaries, documentation and configuration files. In order to avoid the common problem of parallel development on a single source tree, we opt for a solution where we share compiled code (components, libraries …). Every installed module contains the following configuration files: *packageVar* (for compiler settings and paths) and a file *TcshSrc* (for execution settings). To include a module just the respective *packageVar* and *TcshSrc* have to be used. By this way the required compiler and execution settings are inherited. Moreover, our build system gives support for this multi-platform installation tree when compiling and installing modules.

F. *Creation of System Graph*

At this step of the implementation phase the communication hypothesis and the implemented components are used to build the system graph. In order to improve the design and to simplify the creation of our applications we have developed DTBOS (Design Tool for Brain Operating System).

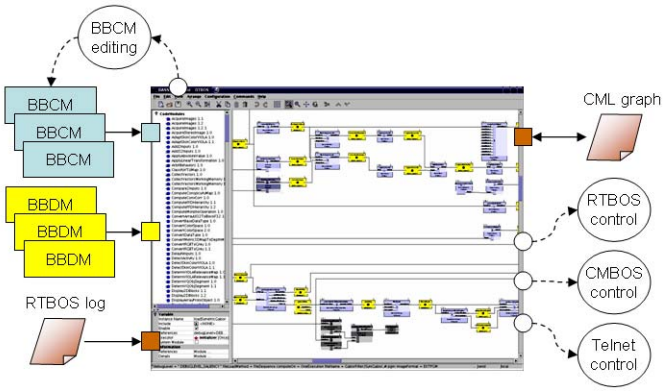

Figure 3: **Design environment DTBOS. White circles identify planed extensions of the tool.**

DTBOS is a RAD-like development environment that allows importing RTBOS Modules (CD/DM) into a toolbar with which users can draw applications in a graphical way. CM/DMs are drawn like boxes with input/output ports; connections can be created by just dragging links from one port to another. DTBOS can read and save files in the same script format (CML, C Macro Language) accepted by RTBOS.

Moreover, with DTBOS we can debug our applications through a playback functionality that reads an RTBOS log session and shows, through an animation, the executed modules step by step. A diagram of DTBOS is shown in Figure 3.

G. *Run, Monitor and Debug*

The last step of the implementation phase consists of the execution of the system, the debugging and the test in simulated and real-world conditions. The following tools support this step:

Distributed middleware for real-time applications: we have created RTBOS (Real-Time Brain Operating System), a distributed middleware for multi-platform real-time modular applications. The requirements from the beginning were to provide in integration environment that is suitable for vision, audition, internal prediction and decisions as well as behaviour generation and control of actuators. The sensory streams demand for a high data bandwidth, the internal processes for a high connectivity and the control aspects for a low latency and a strict determinism in processing and communication. The internal mechanisms account for those requirements by providing low overhead communication and data sharing mechanisms as well as deterministic thread control. An RTBOS application is defined by a set of: Computing Modules (CM), Data Modules (DM), a Connectivity Graph and Execution Patterns. A CM is an RTBOS Module that encapsulates a BBCM component; a DM is an RTBOS Module that encapsulates a BBDM component; a Connectivity Graph defines the connections between CM and DM while an Execution Pattern determines the ways CMs are executed. We have identified several patterns, which can be defined for arbitrary sub-partitions of the system instance.

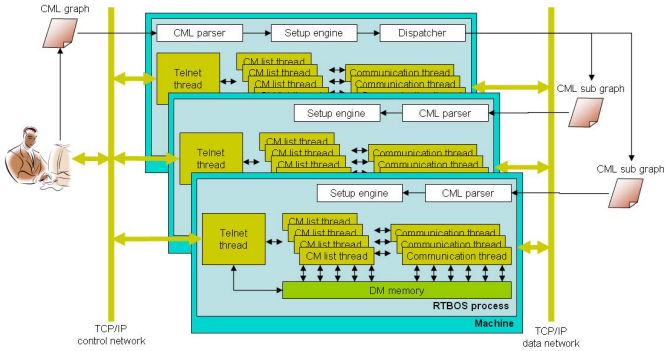

Figure 4: Run-time environment RTBOS.

The major Execution Patterns are: *Parallel* – which executes each CM in a separated thread. This execution pattern is necessary for executing parts of the system in parallel with each other. It is the basic execution pattern since our underlying hypothesis is an asynchronous system that can be synchronized if necessary. *Sequential* – which executes all CMs in a sequence, all in one single thread. The choice of the

execution pattern allows for a fine grained control of the computing resource allocation.

When RTBOS starts an application, it loads the CM/DM modules defined in the CML script file; connects them following the Connectivity Graphs, initializes all modules and then executes the full application (Setup engine in Figure 4). The parallelism of an RTBOS application may employ threads (within a machine) or processes (across machines). RTBOS, through the concept of Execution Patterns simplifies drastically the design and the creation of parallel applications. Moreover, RTBOS handles automatically data allocation, communication and synchronization between threads and processes even across computer boundaries.

The script file defining the applications follows a simple language definition providing a set of commands for managing applications as described above. The script file is plain text and can be quickly edited by any text editor. Since it is following a language specification it can be parsed and produced by other tools like the design environment DTBOS. This feature leads to a higher acceptance for both researchers preferring to work with configuration files and researchers preferring GUIs.

Monitoring control system: we have created also the monitoring system CMBOS (Control Monitor system for Brain Operating System). CMBOS is a demon process that runs in a monitoring machine. This demon, from one side accepts requests for data monitoring, while on the other side opens automatically the appropriate user interface to show the requested data to be monitored. CMBOS contains a set of already defined user interfaces that show 2D graphs, counters, images and others types of visualization tools. Users can implement new visualization tools and new data types to expand the functionality of CMBOS.

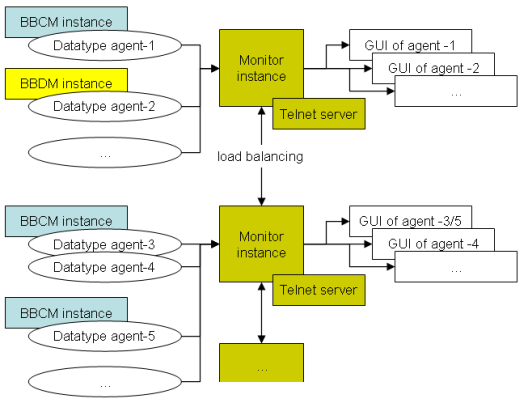

Figure 5: Control and monitoring system CMBOS.

H. *Closing the Loop*

Base on the described overall process we have now introduced all tools for supporting our work. In close collaboration with the researchers we are continuously working on the refinement and the extension of the covered concepts. We have already started the integration of all the mentioned tools into one single environment. Our target is to make DTBOS the central interface, the only tool that users would need to know. From DTBOS it will be possible to create new BBCM/BBDM components, edit them, compile them, import them automatically in the toolbar, use them to draw applications, execute the RTBOS session and open CMBOS monitoring sessions directly through the DTBOS interface (see Figure 3).

All our tools have been designed to work on several platforms. We currently support Linux, Sun Solaris, Windows (native and CygWin) and VxWorks. We are interested in expanding the list of supported platforms in order to improve stability, portability and performance.

III. Our Applications and Experiments

The tools reported on in this paper have been growing together with our scientific research systems over the last four years. The valuable feedback from the researchers concerned with their own functional methods and algorithms has found its way directly into the conceptual improvements of the tools. The tools, especially RTBOS have been tested on benchmark problems in order to examine the theoretical performance. As a result of these experiments we could estimate the overhead introduced by RTBOS compared to a hand coded system. It is approximately one percent of the overall processing time, assuming extremely simple processing elements spending very little time on actual computation. This assumption represents a theoretical worst case setting for middleware systems.

We will now report on the performance of the tools in real-time real world settings. There are three major areas of applications and experiments to report on, others are subject to current research with publications under preparation.

A. *Interactive Vision System*

The first area is concerned with an interactive vision system composed of: image acquisition, saliency computation (from several cues: colour, intensity, saturation, motion, disparity), visuo-motor mapping & learning, gaze selection, object recognition and learning and camera head control.

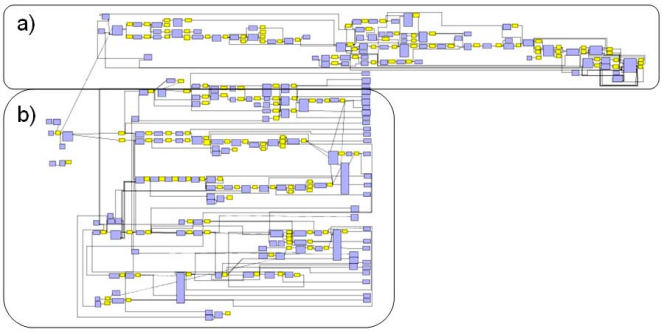

Figure 6: DTBOS graph of our current active vision system. a) Online object recognition and learning sub-system. b) gaze control sub-system.

This system is able to accomplish the following tasks: visuo-motor mapping learning, interactive attention modulation, object fixation and recognition, scene

exploration, simple feature based object search. Some of the corresponding work has been published in [17], [18], [19]. In total 13 researchers have been contributing to and working with this system over the past four years in an incremental fashion.

The system has a brain-like systems architecture, the current connectivity graph with the respective subsystems is shown in Figure 6. It has now been growing over the last four years, reaching a number of 202 BBCM instances as processing modules and 157 BBDM instances as data modules. The average number of instances per coded BBCMs is around two, while the average number of instances per coded BBDMs is five. Those numbers are a mild estimate for the reuse of components within one systems instance. There are several variations of the system with at least six current computation threads up to a fully parallel system with the number of threads in the order of the number of processing modules. The execution time for the gaze selection computation cycle is in the order 200ms, the time for the online learning between 80 and 320ms, and the time for the actuator control is between 500ms. The current version runs distributed across several CPUs located on two SMP machines connected via dedicated GigaBit Ethernet.

For this system clearly the connectivity / communication and the number of processing modules to manage are the challenges for software environment, less the latency constraints for the processing.

B. *Binaural Sound Localization System*

More emphasis on the latency constraints is put in the second area, which is concerned with a real-time binaural sound localization system. The system performs the estimation of the pan angle of incoming sound events and the control of the gaze direction of a robot head based on this estimation in real-time. It is composed of the following stages: sound acquisition; 3 streams for sound position computation (IIT, IID and IED); integration stage and head control. The corresponding work is being published in [20] and [21].

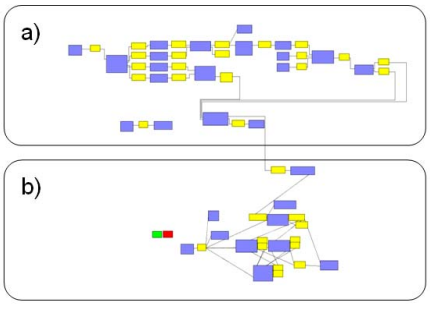

Figure 7: **DTBOS graph of our visually guided whole body interaction system. a) vision sub-system. b) behaviour control sub-system.**

The main two subsystems are the *hardware interface* and the *sound localization computation*. The hardware interface performs the stereo acquisition of the sound streams and the control of the ASIMO head, while the sound localization computation performs a parallel estimation of the sound source pan angle. The stream cycle is 50ms for both subsystems and the number of threads is ten. The total number of BBCM instances is 222, and the total number of BBDM instances is 275. Again, the two subsystems are executed distribute across several CPUs located on two SMP machines connected via dedicated GigaBit Ethernet. The connectivity graph of the system is sketched in Figure 8. In total seven researchers are contributing to this system.

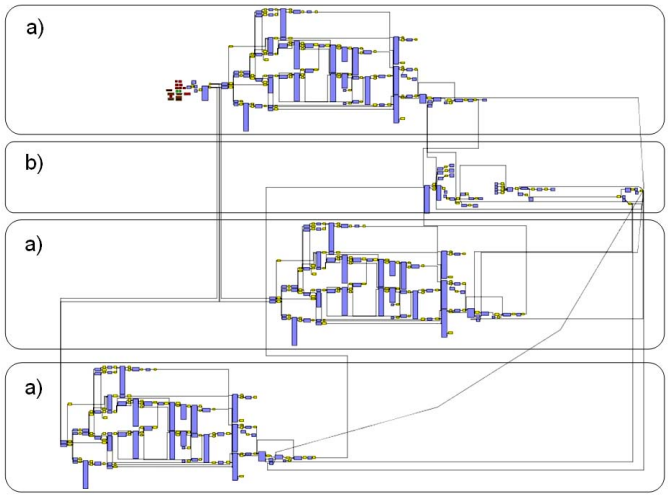

Figure 8: DTBOS graph of our real-time binaural sound localization system. a) sound localization computation parallelized on three processors computing 30 frequency channel each. b) the hardware interface.

While the previously presented vision system is very asynchronously oriented, the auditory system is more synchronous. The reason is that the visual setting is from the signal type more continuous allowing for a more loose kind of synchronization, while the auditory setting is composed of rare events that have to be tightly synchronized in order to meaningfully fuse the different analysis streams into one stable pan angle estimation. Both kinds of processing paradigms could be equally well implemented by the proposed tools.

C. *Visually Guided Whole Body Interaction System*

The third area is concerned with a visually guided whole body interaction system for our robot ASIMO. The system allows ASIMO to gaze, looking for close objects, selecting one and trying to approach the selected target by whole body motions. Parts of this work are being published in [22] and [23]. This system is composed by three main sub-systems: *ASIMO*; a *vision* sub-system and a *behaviour control* sub-system. The vision sub-system receives images form the ASIMO stereo camera and creates a set of possible targets. The control sub-system selects one target and then controls the behaviours that ASIMO has to execute. Behaviours are currently: walking, grasping and tracking. The vision subsystem has a stream cycle of 150ms, the control system has

a stream cycle of 5ms. The total number of instantiated BBCMs is 29, the total number of instantiated BBDMs is 35. The connectivity graph is sketched in Figure 7.

The system is distributed across three different machines. The mixture of environment sensing and systems control puts the hardest constraints on the tools, especially on RTBOS.

Additional to the experience within the three different fields there are some general observations we would like to report on now. We could experience that the limits of our tools are reached if we are close to the limits of the underlying operating systems and hardware. Those limits are mainly due to general purpose schedulers and limited network bandwidth. For us this is a mild indicator that the overall overhead introduced by our tools is small.

Major parts of our systems have been contributed by non-systems programmers. With the aid of the offered tools they were able to create and manage real-time multi-threaded parts of the systems.

The overall reuse of components across the different fields depends on the level of granularity of the components. Generally speaking, the reuse is higher for fined grained components like simple arithmetic functions on the inputs. For example, there is a large overlap of the simple components between the visual components of the first and the third area, but exact numbers have to be determined.

The chosen decomposition of the systems as well as the means for working visually and monitoring various systems properties allows for real research on systems level. As pointed out in the paragraph on the auditory systems different kinds of processing and communication paradigms can be created and managed with the same tools. For example, in the area of the active vision system we could investigate different synchronization mechanisms without touching the internals of the processing modules.

IV. Summary

In this paper we described the research and development process that allow us to incrementally improve our real-time distributed embodied intelligent system hypotheses and in parallel progress with our integrated software development environment. During the last four years we gained experience integrating such systems and the design decision we took during this process allowed us to quickly and reliably develop complex interactive systems. We are now confident that such approach is mandatory to deal with increasing complexity.

Acknowledgment

The work presented in this paper is the result of a tight cooperation among our researchers. Work that would have never been the same without the doubts of Heiko Wersing, the bugs found and created by Tobias Rodemann, the curiosity of Inna Mikhailova, the patients of our students and the support and technical skills of our external contractors.

References

[1] Antonello Ceravola and Christian Goerick, "An Integrated Approach Towards Researching and Designing Real-Time Brain-Like Computing Systems", submitted on The First International Symposium on Nature-Inspired Systems for Parallel, Asynchronous and Decentralised Environment, 2006, Bristol, England.

[2] Frank Lüders, "Adopting a Software Component Model in Real-Time systems Developments", Proceedings of the 28th Annual NASA/IEEE Software Engineering Workshop, IEEE Computer Society Press, 2004.

[3] Matthias Scheutz, Virgil Andronache, "Architectural Mechanisms for Dynamic Changes of Behavior Selections Strategies in Behavior-Based Systems", Systems, Man and Cybernetics, Part B, IEEE Transactions, 2004.

[4] Richard N.Langlois, "Modularity in Technology, Organization, and Society", University of Connecticut - Department of Economics, 1999.

[5] Peter Carruthers "Moderately massive modularity", in A. O'Hear (ed.), Mind and Persons, Cambridge University Press, 2003

[6] Brian Gerkey, Richard T. Vaughan and Andrew Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems", in 11th International Conference on Advanced Robotics, Coimbra, Portugal (ICAR'03), 2003

[7] Giorgio Metta, Paul Fitzpatrick and Lorenzo Natale, "YARP: Yet Another Robot Platform", International Journal on Advanced Robotics Systems, 2005.

[8] Real-Time Innovation, "ControlShell User's Manual, Version 7.0", Real-Time Innovation Inc., California, 2001.

[9] Issa A.D. Nesnas, "CLARAty: Towards Standardized Abstractions for Robotic Systems", workshop on Principle and Practice of Software Development in Robotics, ICRA 2005, Barcelona, Spain, 2005.

[10] Christian Schlegel, "A Component Approach for Robotics Software: Communication Patterns in the OROCOS Context", workshop on Principle and Practice of Software Development in Robotics, ICRA 2005, Barcelona, Spain, 2005.

[11] Kazuo Tanie, "Standardization of Robotic Components and Future Robotics Business", IROS 2004, AIST – Japan, 2004.

[12] Olivier Stasse, Yasuo Kuniyoshi, "PredN: Achieving efficiency and code re-usability in a programming system for complex robotic ap-plications", International Conference on Robotics and Automation (ICRA), San Francisco, CA, USA , 2000.

[13] http://www.intel.com/cd/software/products/asmo-na/eng/perflib/index.htm.

[14] http://www.sun.com/processors/vis/mlib.html.

[15] http://svnbook.red-bean.com.

[16] http://www.gnu.org/software/make/manual/make.html.

[17] Tobias Rodemann, Frank Joublin and Edgar Körner, "Saccade Adaptation on a 2 DoF Camera Head", Horst-Michael Groß and Klaus Debes and Hans-Joachim Böhme, Fortschrittsberichte des VDI, 2004.

[18] Christian Goerick, Heiko Wersing, Inna Mikhailova and Mark Dunn, "Peripersonal space and object recognition for humanoids", Proceedings of the IEEE/RSJ International Conference on Humanoid Robots (Humanois 2005), Tsukuba, Japan, 2005.

[19] S. Kirstein, H. Wersing and E. Körner, "Rapid Online Learning of Objects in a Biologically Motivated Recognition Architecture", in 27th Pattern Recognition Symposium DAGM, 2005.

[20] Martin Heckmann, Tobias Rodemann, Frank Joublin, Christian Goerick and Björn Schölling, "Auditory Inspired Binaural Robust Sound Source Localization in Echoic and Noisy Environments", submitted in Proceedings of the International Conference on Intelligent Robots & Systems (IROS), 2006.

[21] Tobias Rodemann, Martin Heckmann, Björn Schölling, Frank Joublin and Christian Goerick, "Real-time sound localization with a binaural head-system using a biologically-inspired cue-triple mapping", submitted in Proceedings of the International Conference on Intelligent Robots & Systems (IROS), 2006.

[22] Micheal Gienger, Herbert Janßen and Christian Goerick, "Task-Oriented Whole Body Motion for Humanoid Robots", Proceedings of the IEEE/RSJ International Conference on Humanoid Robots (Humanois 2005), Tsukuba, Japan, 2005.

[23] Micheal Gienger, Herbert Janßen and Christian Goerick, "Exploiting Task Intervals for Whole Body Robot Control", submitted in Proceedings of the International Conference on Intelligent Robots & Systems (IROS), 2006.