An Architecture for Real-Time Control in Multi-Robot Systems

Daniel Althoff, Omiros Kourakos, Martin Lawitzky, Alexander Mörtl, Matthias Rambow, Florian Rohrmüller, Dražen Brščić, Dirk Wollherr, Sandra Hirche and Martin Buss

Abstract This paper presents a novel robotic architecture that is suitable for modular distributed multi-robot systems. The architecture is based on an interface supporting real-time inter-process communication, which allows simple and highly efficient data exchange between the modules. It allows monitoring of the internal system state and easy logging, thus facilitating the module development. The extension to distributed systems is provided through a communication middleware, which enables fast and transparent exchange of data through the network, although without real-time guarantees. The advantages and disadvantages of the architecture are rated using an existing framework for evaluation of robot architectures.

1 Introduction

Software complexity of the emerging generation of versatile robotic systems increases alongside with their capabilities. Cooperative action of multiple robots, each controlled by numerous software modules, requires seamless message and data exchange internally among the modules, among the robots as well as with distributed sensor systems.

We consider a scenario of multiple robots operating in a populated environment and interacting with humans. Multiple sensors such as tracking systems, on-board laser range finders and force/position sensors permanently generate new data. Data-processing modules such as localization and the generation of a 3-D world representation retrieve sensor data and update the world model. This in turn is utilized by high-level reasoning and planning algorithms in order to issue appropriate plans and commands. Finally, low-level processes control actuators in order to execute these commands.

The authors are with the Institute of Automatic Control Engineering (LSR) of the Technische Universität München, D-80290 München, Germany {da, omiros.kourakos, ml, moertl, rambow, rohrmueller, drazen, dw, hirche, mb}@tum.de

To cope with the challenges arising from this kind of systems sophisticated software frameworks that provide interfaces between the modules have been developed. Popular examples such as *Player* [1] or *ROS* [2] are user-friendly frameworks incorporating a large number of open-source modules which enable a rapid implementation of robotic applications. Even though these software frameworks offer mature basic services in various respects, so far they cannot fully satisfy all requirements of highly modularized and distributed next-generation cognitive robotic systems like:

- support of feedback control under real-time constraints,
- seamless data acquisition and sharing among multiple modules in one robot,
- inter-connectivity and bandwidth for efficient distributed operation,
- elaborate structure to provide maintainability and scalability, and
- user support tools that convey the system state in an intuitive manner and facilitate debugging and data logging.

In this paper we present a software architecture suited for research on interaction and cooperation in complex multi-robot scenarios. The architecture focuses on distributed robotic systems and provides support from real-time execution in low-level control up to efficient high-level information exchange.

This paper is organized as follows. Section 2 presents the proposed architecture and gives a qualitative evaluation that illustrates its strengths and weaknesses. In section 3 a brief application example is given. Finally, section 4 concludes the paper.

2 The *ARCADE* framework

ARCADE stands for **A**rchitecture for **R**eal-time **C**ontrol and **A**utonomous **D**istributed **E**xecution. It is a data-driven architecture built up on top of a real-time capable interface for inter-process communication and the IceStorm publisher-subscriber extension of the ICE RPC framework [3]. The *ARCADE* framework is illustrated in Fig. 1 and explained in the following.

2.1 System description

*Real-time database (RTDB):*The real-time database *KogMo-RTDB* [4] – originally developed for autonomous cars – is the central part of the framework. This real-time database is not a database in its traditional meaning, however it implements a number of features of hierarchical databases. The RTDB provides real-time guarantees for management and exchange of data structures, defined as data objects in the RTDB terminology. It handles all local inter-process communication conveniently, allows record/replay of states, buffers data and acts as a powerful central hub for all data and command transactions. A further core functionality of the RTDB is the maintenance of hierarchical data objects. Data objects can be easily inserted, updated, deleted and attached as child objects. Searching and reading can be executed

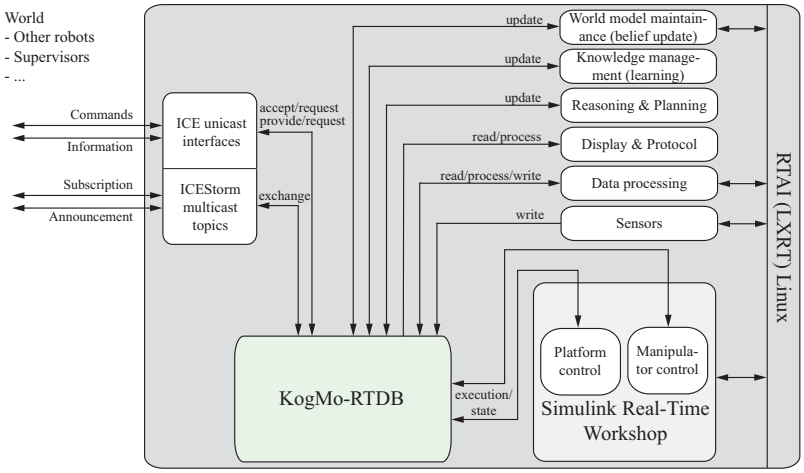

Fig. 1 Overview of the *ARCADE* framework.

in a blocking or non-blocking fashion, depending on the specific needs. In addition, circular buffers of configurable size are maintained to hold histories of data objects. The RTDB offers very high convenience and ease of use for managing complex and large amounts of data while giving real-time assurances. Performance measurements for the database back end throughput and jitter are given in [5], for example are the average/worst case times in a strongly busy system without real-time configuration $23\mu s/181681\mu s$ for write operations and $17\mu s/10721\mu s$ for read operations respectively. With real-time configuration the average times are similar while the worst case times are strongly reduced to $134\mu s$ (write) and $62\mu s$ (read) [5].

Accessing the RTDB: The RTDB can be accessed using the available *ARCADE* interfaces. An interface is a C++ class implementing a set of methods that operate on a data object. Additionally, the *ARCADE* interfaces automatically inherit the RTDB interface methods (read, write, search etc.).

Any process that uses one or more *ARCADE* interfaces is defined as a module. For illustration we describe one producer and one consumer module that use the same interface. These modules could be a driver for a laser range finder and a line extraction algorithm respectively. In this case the interface comprises the data object, which is an array of floating point numbers holding the range measurements, and methods to set (setData) and to get (getData) the data object.

Both modules first have to create a connection to the RTDB. The producer has to instantiate an *ARCADE* interface and insert it in the RTDB with a descriptive name. This name can be used by other modules to search and subscribe to this data. The setData method is used to update the local copy of the data object which subsequently is committed using the write method. The RTDB itself imposes no constraints on the update rate (read/write) of the module which solely depends on the refresh rate of the laser range finder.

The consumer has to instantiate the same interface and associate it with the existing object by searching the RTDB with the descriptive name as explained above. The local copy is updated using the read method and the getData method to access the data object and process it. Both modules can be implemented either as best-effort (i.e. non real-time) processes or real-time tasks using RTAI/LXRT [6].

In addition, the *ARCADE* framework provides library blocks for Simulink to access the RTDB. This enables rapid-prototypical control design in Simulink together with Real-Time Workshop [7] using the RTAI target.

Inter-RTDB communication: In order to transfer data between RTDBs the ZeroC/ICE middleware [3] is integrated into the *ARCADE* framework. ICE provides an operating system independent interface and supports the most common object-oriented programming languages. Furthermore it allows type safe and clear interface definitions. ICE servers give access to any RTDB within the entire system for reading data and setting command objects.

For information exchange among the robots, the IceStorm multicast extension of ICE is used. IceStorm is a publisher/subscriber mechanism for unidirectional information exchange. Any module is able to create and publish topics via IceStorm. Agents on the network can subscribe to topics and receive automatic updates on content changes. In our wireless network setup, bandwidth efficiency is a crucial factor. For the distribution of high-frequency sensor data we extended the IceStorm multicast to allow the specification of a desired update rate alongside with topic subscriptions. The maximum desired update rate for each topic determines the rate of multicast frames sent through the network. Hence, an efficient synchronization of robots in order to maintain a global up-to-date world model is possible.

In the *ARCADE* framework inter-RTDB data exchange can be implemented with corresponding ICE interfaces. However, the abstraction of this procedure remains future work.

Command exchange via the RTDB: The RTDB was originally designed to distribute data streams rather than commands. Straightforward updating of a command object in the database is not possible in case command reception has to be guaranteed. The database command object could be updated more frequently than it is read by the receiver. Due to the finite length of the object history maintained by the RTDB successful transmission cannot be guaranteed and as a consequence commands can be lost. In this respect, to enable the command exchange via the database a three-way handshake – referred to as “postbox system” – has been integrated based on the functionalities provided by the RTDB. It is illustrated in Fig. 2.

A postbox object (P) is created by the receiver side (RX) and then regularly checked for new message objects. A sender process (TX) can then insert a message object, which is attached to the postbox. During its next update cycle the receiver side reads all new messages, which in turn get an acknowledgement object appended. Since only the process which created a specific object obtains write access to it, the sender process regularly checks all its previously sent objects and deletes them as soon as a corresponding acknowledgement object is detected. Consequently this three-way handshake obviates memory leaks and ensures reception feedback for both sides. Instead of the acknowledgement object the RX can also

append any arbitrary response object which may contain already the data requested by TX.

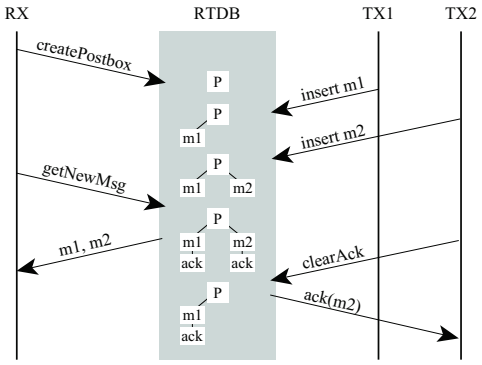

Fig. 2 Scheme of the *ARCADE* postbox system.

2.2 Evaluation of the robot architecture

In order to further highlight the supported features of *ARCADE* we evaluated it using the conceptual framework for comparing robot architectures introduced in [8]. Other frameworks for architecture comparison were proposed, e.g. in [9]. However, [8] was chosen because of the detailed and specific description of the evaluation criteria and rating method.

Table 1 shows the evaluation criteria and the corresponding level of support in *ARCADE*. A detailed description of all the criteria is given in [8]. As opposed to the original framework we do not consider criteria regarding algorithm implementations such as localization or route planning in our evaluation.

Architectural primitives (F1.1) refer to predefined functional and/or knowledge primitives. *ARCADE* provides fundamental components to control the robotic hardware and to perform a set of basic actions which can be scheduled via a priority-based action controller. However, a kind of generic behavior framework is not provided resulting in a *somewhat supported* mark. With respect to the software engineering criteria (F1.2) *ARCADE* provides coding guidelines and a set of basic classes enabling code reusability and simplification. Nevertheless an explicit theoretical foundation is missing. Architecture neutrality (F1.3) is not provided since the presented framework belongs to the class of blackboard architectures.

Since the RTDB only supports Linux the same applies also to *ARCADE* (F2.1). Additionally besides our own robotic hardware only few further devices – such as the SICK LSM200, S300 or the JR3 force/torque sensor – are currently supported (F2.2). Regarding the simulator (F2.3), *ARCADE* makes use of several different simulation/visualization tools, such as the *ARCADE* Visualizer, see Fig. 3, which

Category	Criteria	*ARCADE*
Specification	F1.1 Architectural Primitives	$\odot$
F1	F1.2 Software Engineering	$\odot$
	F1.3 Architecture Neutrality	no
	F2.1 Operating System	Linux
Platform	F2.2 Hardware Support	$\ominus$
F2	F2.3 Simulator	$\odot$
	F2.4 Configuration Method	$\oplus$
	F3.1 Low-level Communication	ICE,RTDB
	F3.2 Logging Facilities	$\oplus$
	F3.3 Debugging Facilities	$\oplus$
Infrastructure	F3.4 Distribution Mechanisms	$\oplus$
F3	F3.5 Scalability	$\odot$
	F3.6 Component Mobility	$\odot$
	F3.7 Monitoring/Management	$\oplus$
	F3.8 Security	$\odot$
	F3.9 Fault-tolerance	$\odot$
	F4.1 Programming Language	C++, Simulink
	F4.2 High-level Language	no
Implementation	F4.3 Documentation	$\odot$
F4	F4.4 Real-time Operation	yes
	F4.5 Graphical Interface	$\odot$
	F4.6 Software Integration	$\ominus$

Table 1 Qualitative evaluation of the *ARCADE* robotic architecture: $\oplus$ = *well supported*, $\odot$ = *somewhat supported*, $\ominus$ = *not supported*. The criteria codes, names and specifications were taken from [8].

provide the means for dynamic or multi-robot simulations but are currently separate modules and not yet fully integrated into a single simulator. From the configuration methods point of view (F2.4), XML-files are supported. Furthermore graphical interfaces are provided to set and modify parameters online and to send commands to the respective modules.

The low-level communication (F3.1) through RTDB and ICE was described in section 2.1. The RTDB provides a record and replay functionality with real-time support, i.e. data can be captured of part of or all the database objects and later be replayed keeping the original timing (F3.2). Additionally, console output of any module can be remotely captured and stored in files. Due to the distributed processing, any module – except the central RTDB – can be suspended, modified and restarted during runtime which facilitates the debugging (F3.3). Nevertheless no automatic mechanism keeping care of continuing system operation is provided, which would be required for well supported component mobility (F3.5). As already mentioned, the distribution mechanisms (F3.4) are well supported through the use of ICE and IceStorm. The scalability (F3.5) as defined in [8] is dependent on the values of the other criteria and results in our case in a *somewhat supported* mark.

The *ARCADE Inspector* (Fig. 3) is a sophisticated graphical interface to start/stop modules, view the current state of any database object or send commands providing a convenient tool for monitoring and management (F3.7). Security (F3.8) can only be indirectly supported through usage of the SSL protocol in ICE, and the RTDB

support for the single producer concept which allows data modification of an object only by its source process. The distributed processing nature of *ARCADE* makes it fairly tolerant to failures (F3.9), but its notable weaknesses are the dependency of all subscribed modules to their local RTDB and lack of active failure recovery.

ARCADE provides support for C++ and Simulink (F4.1). Even though an interface for high level commands is provided, which could be incorporated into a behavior framework, no explicit high level language is integrated (F4.2). Even though code documentation is available, no API or user guidelines are provided at the moment (F4.3). While none of the compared architectures in [8] provides real-time support (F4.4), it is the main strength and also the crucial motivation for the development of *ARCADE*. While the *ARCADE Inspector* visualizes each module operation, a graphical tool for the design of control code (F4.5) is not provided (except Simulink). A standard API for software integration (F4.6) is *not supported* at all.

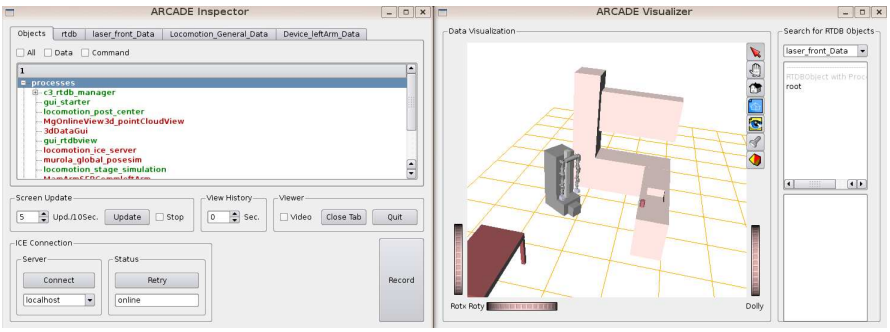

Fig. 3 The *ARCADE* Inspector lists the tree of all database objects, provides access to the object data and a record functionality. The *ARCADE* Visualizer provides a 3D visualization of the current system state.

In order to obtain also a quantitative measure, we calculated the feature score of the reduced criteria list according to [8] using equal weights. For binary-valued criteria a value of 0 or 2, for ternary-valued criteria a value of 0, 1 or 2 is assigned. Accumulating the values in Table 1 leads to a total score of 55%. In a comparison with the architectures in [8] *ARCADE* ranks sixth out of ten. The main weakness of *ARCADE* is the specification category (F1) where it scores 33%, which is below all other architectures. This follows from the initial unavailability of components (F1.1 and F1.2), such as standard hardware drivers, integrated algorithms and clearly structured interfaces which in general are provided by open-source frameworks. The Linux-only support (F2.1) may be regarded a further disadvantage, especially when a large number of researchers are involved, and the need for multi-platform support arises.

ARCADE is very strong in the infrastructure category (F3) achieving a score of 75%, where only *ADE* (100%) ranks better [8]. Consequently, while further work is required in order to simplify its portability, *ARCADE* provides very good tools and mechanisms to develop and operate a running system.

Yet, for highly distributed systems considered in this paper real-time support (F4.4), sophisticated management (F3.7) and distribution mechanisms (F.3.4) are mandatory. As opposed to *ARCADE*, other existing architectures do not adequately meet these aspects.

3 Application Example

In order to illustrate the integration of modules and the usage of the *ARCADE* framework, an example of a fast reactive collision avoidance application for robotic arms based on virtual forces is described. The example application comprises several interconnected modules which are distributed on two computers as shown in Fig. 4, each running an instance of the RTDB. A potential reason for using multiple computers is the distribution of computational load.

The world-builder module (WB) running on computer 1 processes the raw data stream of the sensor module and generates a world model that includes a list of objects and their respective collision shape primitives. This model is mirrored from RTDB 1 to RTDB 2 using the inter-RTDB communication mechanisms.

The world observer (WO) running on computer 2 monitors changes of the world model and passes a list of relevant world collision shapes to the force generator module (FG). The FG calculates virtual forces F_v acting on the robotic arms based on the current state $(q,\dot{q})$ of the arms and the current pose of the platform.

The update frequency of the world model depends on the kind of sensors and processing algorithms used, but typically is less than 50 Hz. However, the FG is running at the same frequency as the actual control loop, which is typically about 1 kHz.

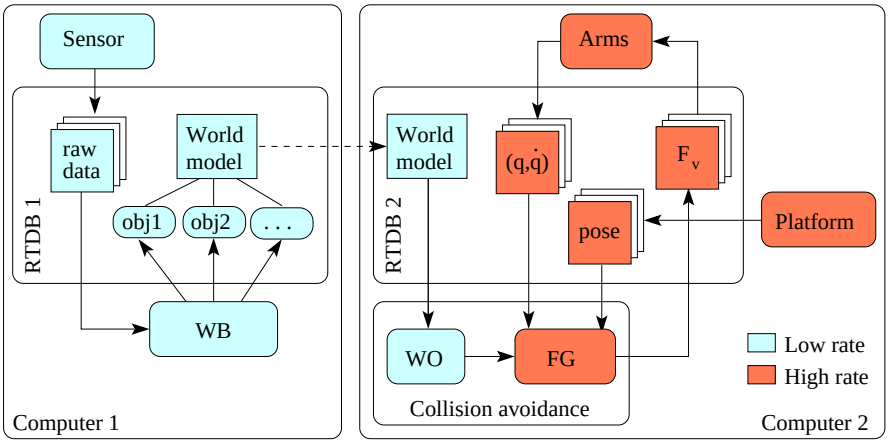

Fig. 4 Overview of the collision avoidance application: The *ARCADE* framework manages distributed computing and different update rates of the modules.

4 Conclusion

This paper presents *ARCADE*, a data-driven robot architecture combining KogMo-RTDB and the ICE communication middleware. The RTDB provides a central element of *ARCADE* and an easy way for exchange of data between the modules, whereas ICE and IceStorm provide the connection to distributed modules and other RTDBs. The architecture is able to provide both real-time guarantees for low-level robot control and a simple and effective information exchange between distributed modules in multi-robot systems. The presented evaluation showed that *ARCADE* gives a very good solution for distributed multi-robot systems, but still has several weaknesses, mostly due to unavailability of components. The ease of use and versatility of the architecture was illustrated in an application example.

5 Acknowledgements

We would like to thank Matthias Goebl from the Institute for Real-Time Computer Systems of TU München for allowing us access to the KogMo-RTDB, its documentation and accompanying programs which served as a starting point for our architecture.

This work is supported in part within the DFG excellence initiative research cluster Cognition for Technical Systems - CoTeSys (`www.cotesys.org`).

References

1. B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *In Proceedings of the 11th International Conference on Advanced Robotics (ICAR '03)*, (Coimbra, Portugal), pp. 317–323, June 2003.
2. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. B. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *International Conference on Robotics and Automation*, Open-Source Software workshop, 2009.
3. M. Henning, "A new approach to object-oriented middleware," *IEEE Internet Computing*, vol. 8, no. 1, pp. 66–75, 2004.
4. M. Goebl and G. Färber, "A real-time-capable hard- and software architecture for joint image and knowledge processing in cognitive automobiles," in *Proceedings of the 2007 IEEE Intelligent Vehicles Symposium*, pp. 734–740, June 2007.
5. M. Goebl, *Eine realzeitfähige Architektur zur Integration kognitiver Funktionen*. PhD thesis, Technische Universität München, Institute for Real-Time Computer Systems, 2009.
6. L. Dozio and P. Mantegazza, "Real time distributed control systems using RTAI," in *Proceedings of the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, (Hakodate, Japan), May 2003.
7. The Mathworks, *Real-Time Workshop 7 - Generate C code from Simulink models and MATLAB code*, 2007.
8. J. F. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007.
9. A. Shakhimardanov and E. Prassler, "Comparative evaluation of robotic software integration systems: A case study," in *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (San Diego, CA, USA), pp. 3031–3037, 2007.