Knowledge based agent architecture for high level control of mobile robots

Gottfried Koppensteiner and Munir Merdan
Vienna University of Technology, Institute for Automation and Control
koppensteiner@acin.tuwien.ac.at, merdan@acin.tuwien.ac.at

Clemens Koza, Christoph Krofitsch, Manuel Parg and Antonio Prvulovic
Vienna Institute of Technology, Department for Information Technology
{clemens.koza, christoph.krofitsch, m.parg, antonio.prvulovic} @gmx.at

Knowledge based agent architecture for high level control of mobile robots

1. Introduction

Disassembly has become a vital industry process due to the increasing necessity of optimizing resource usage. Currently, disassembly processes are partially automated only in a small set of cases such as: single use cameras [9], PCs [13], printed circuit boards as well as LCD monitors [4]. The main limitation factors are: non-uniformity of returned product models creating great uncertainty in the system control and structural configuration; questionable economic benefits due to the small automation grade; as well as custom developed character of current implementations, meaning that costs can be spread neither over a broad range of applications nor over time [3]. Besides, the rigid character and weak adaptation capabilities of the currently implemented solutions which have centralized and hierarchical control structures, limit their ability to respond efficiently and effectively on dynamic changes. Mobile robots offer new possibilities, e.g. for flexible disassembly [6]. The state-of-the-art of mobile robot technology and predictions of future development are giving a clear view that mobile robots are going to be an essential part of every manufacturing process in future [1]. Mobile robots are giving flexibility to the system and increase dynamics of the whole process. However, one of the obstacles in a wider adoption of the mobile robots in the industry is the cost of engineering the robot into the system. The robotics area should develop in such way to reduce the programming requirement and to increase the flexibility of mobile robots for different tasks [8]. In addition, mobile robots for disassembly should be (a) intelligent in the sense of path planning and able to communicate with other robots, (b) cooperative with other (stationary or mobile) robots, and (c) able to form a disassembly multi agent system, which is one of the future possibilities for reducing disassembly costs. Moreover, in order to avoid difficulties in communication in a heterogeneous robot environment, where each robot has its own kinematic structure and programming language, etc., it is necessary to develop standardized communication protocols and methods [6].

To cope with these requirements, we propose a knowledge-intensive multi-agent robot system, which enables ontology-based communication and cooperation among a set of autonomous and heterogeneous units – agents. In our system, each agent supervises one particular mobile robot and, related to the robot's skills, is having its own objectives and knowledge. In this context, ontologies allow the explicit specification of an agent's domain of application, increasing the level of specification of knowledge by incorporating semantics into

the data and promoting knowledge exchange between agents in an explicitly understandable form [10]. An ontology based product model is used [5] to link product designs, disassembly planning and scheduling processes, as well as required disassembly equipment, possessed by a particular mobile robot, in a way that enables automatic reasoning as well as wide data integration. Consequently, on the one side, a vision system can use this model to reason about the content of a captured image. On the other side, an agent controlling a mobile robot can extract required disassembly information from this model to select and perform the necessary actions. The architecture is based on agents that have a rule-based behaviour. Rules are considered as if-then statements applied to the knowledge base. The application of this kind of decision-making mechanism supports a knowledge capture in a more modular and explicit way. This paper is structured as follows. Section II introduces the agent architecture and section III the system integration (controller aspects), which is followed by the system implementation. Finally, the paper is concluded in the fifth section.

2. Agent Architecture

To facilitate the design of multi-agent control systems, in a previous work a generic agent architecture [7, 11, 14] was developed. This architecture clearly separates the control software into two layers: the high level control (HLC) and the low level control (LLC). The LLC layer is in charge of controlling the hardware directly. It is responsible for performing all necessary operations in real-time and is based on the IEC 61499 standard [15]. The HLC layer is responsible for more complex tasks such as coordination, monitoring or diagnostic, which might require a longer computation time. In the rest of the paper, we will present the structure and functionality of the HLC adapted for the control of the mobile robots. Figure 1 shows the architecture of the system which provides the HLC for the mobile robots, with focus on the interplay of the used technologies. It is assumed that the robot controller supports network interfaces; otherwise the system can also be used standalone without any intercommunication.

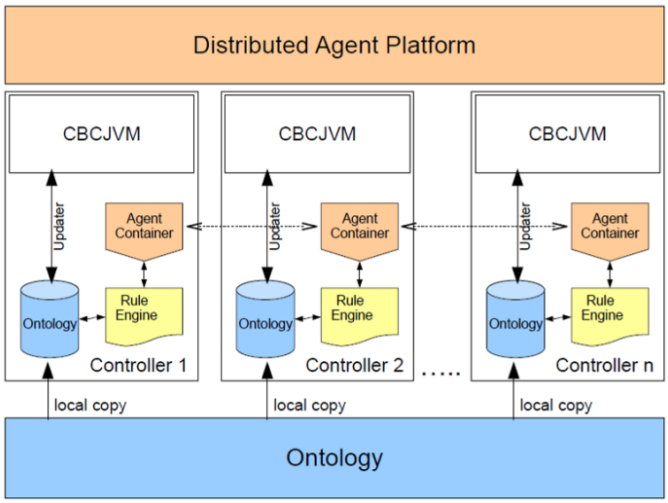

Figure 1: System Architecture

The whole environment is based on one ontology which is copied to each of the controllers. So each controller has components from other controllers (e.g. sensors) in its ontology, but only updates the components of its own robot by itself. The updating process is performed

between the ontology and the specific system-libraries of the controller, in case of the CBCv2 the CBCJVM [2] is used for that purpose. In this process, the controller's own components mapped in the ontology are synchronized with the values from the system-libraries (e.g. sensor values) so that the rule engine can fire appropriate rules for them. These rules are the knowledge base containing the high-level behaviour of the robot, so it's different for each controller. Furthermore, each controller has its own agent container which manages and contains the agents providing the main functionality of the framework. This platform normally runs on one controller which contains the "main container", while the other agent containers have to connect to it in order to join the platform.

3. System Integration

3.1. A LINUX-based low-cost mobile robot controller

The basis of the implemented High Level Robot Control is the CBCv2 [12] which includes an ARM 7 based DAQ/Motor control system, an ARM 9-based CPU/Vision processor running LINUX, an integrated color display as well as a touch screen. It is being used by thousands of middle and high school students in the educational robotics program Botball [16]. Due to the aim of the sparkling science initiative [20], the sponsor of this project, it was one major reason to take a controller system which is well-suited for a collegiate robotics lab. Therefore, the Atmel ARM7 32 bit MCU running at 48 MHz is, because of its speed, availability, and wide range of embedded communication and I/O ports, including 16 10-bit ADCs, good enough to meet the requirements for a robotics Lab. On the other side, the embedded Linux and reloadable firmware provide a complete package for the enhancement with agent and rule based systems. The CBCv2 is a USB host (allowing the use of standard cameras, mass storage and network interfaces) and can also be used as a USB device for software downloads. At the USB port a Wi-Fi Stick can be used; this provides a good possibility for the communication between mobile robots.

3.2. Agent Communication

The agent-based system is implemented within the Java Agent Development Framework (JADE) [17]. As already mentioned in the system architecture, there's the possibility to share ontology values between different controllers. For that matter every controller, particularly the agent container, has to be part of the JADE distributed agent platform which actually runs on one main agent container. After starting, the container determines if he is supposed to run the main container. If so, it gets started, otherwise, the running main container is looked up and connected to. After starting the container (either local or remote) the agents are started and the framework functionality starts, which basically consists of the ontology synchronization process and invocation of the rule engine if the ontology changed.

Additionally, when using intercommunication, other agents are started in order to handle the communication with other controllers. The communication exclusively happens with commands which can be defined by the robot programmer. The framework comes with one already implemented command intended for sharing ontology values: the Query-Command. It uses a simple query language for requesting values from a foreign ontology and updates the

own ontology with the new value when receiving the response. That enables defining rules for foreign ontology values or using knowledge from other robots. Issuing commands such as the Query-Command, can directly be done from rules with the user-defined function "*command*".

4. System Implementation

4.1. Ontology

The framework uses its ontology for multiple purposes. First of all, it serves as the knowledge base for the rule based control system, but additionally, it serves as a place for configuring the robot and the intercommunication. Besides these necessary classes, the ontology may be extended to support application specific needs. Figure 2 shows selected classes from the framework's ontology, some of their attributes and their relationships to each other. The class *CBCController* inherits from *Controller*. While the *Controller* class stores a list of all sensors and actors and is an entry points for future extensions, the *CBCController* adds a more precise view of these, including the software buttons, the acceleration sensor, and the camera.

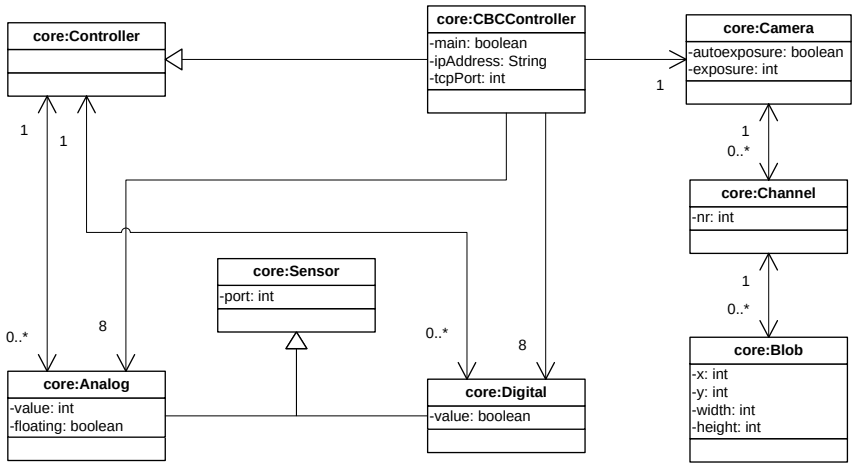

Figure 2: The Ontology

The following code samples will show how rules, implemented in the Jess Language [18] can use different sensors in an environment where multiple controllers are present:

```
(defrule go_on_click
  (Controller (OBJECT ?c) (aButton ?b))
  (Digital (OBJECT ?b) (value TRUE))
  (Motor (OBJECT ?m) (controller ?c) (port 0))
  (test (eq ?c (fetch "controller")))
=>
  (?m moveAtVelocity 1000)
)
```

```
(defrule stop_on_voltage
  (Controller (OBJECT ?c) (powerlevel ?p))
  (Analog (OBJECT ?p) {value < 512})
  (Motor (OBJECT ?m) (controller ?c) (port 0)
  (test (eq ?c (fetch "controller")))
=>
  (?m off)
)
```

Due to memory constraints of the CBCv2, it was not possible to use the Protégé [19] ontology API at runtime. Instead, the OWL-ontology is converted into java objects and stored to a file at build time. The access to the serialized file is then possible on the CBCv2. This reimplementation of the ontology access led to designing a general interface between ontologies and the framework. Future versions can use other technologies for accessing

ontologies by implementing this interface. The mapping is done in two stages: firstly, the classes designed in the ontology are converted into Java classes. Secondly, after the generated classes are compiled, the objects contained in the ontology are read, instantiated, and written to a file via Java serialization. At runtime, the objects are read into the framework via the ontology interface, and added to the rules engine's fact base.

4.2. Interfaces

Modify Ontology: The ontology, i.e. the owl-file, can be extended with user defined classes and instances. Therefore two things are important. At first there are special prefixes in the ontology used to determine to which part of the ontology the class belongs. In the ontology of this framework there are currently prefixes such as "core" and "cbc" in use. The second important notice is that the properties in the ontology have a special naming convention. The name of a property has to be: "<classname>_<propertyname>". The convention was designed because every property name must be unique in the whole ontology.

Change ontology type: In the current state of the framework every class defined in the ontology will be mapped as interface and implementation as Java source. If the ontology requirements are changing or the framework is running on another controller, which has more performance, it would probably be better to use another ontology mapping (e.g. reading directly from the OWL File).

Create Agents: The framework can be extended by adding new agents. At first there must be the agent itself, which inherits *jade.core.Agent*. Every agent needs at minimum one *Behaviour*, where the functionality of the agent is stored. According super classes are *CyclicBehaviour* and *OneShotBehaviour*.

Add Commands: The framework can be extended by adding new commands which can be used by calling the Jess Userfunction *command* in the right-hand side of a rule. Basically there are three steps of writing a new command: implementing the command, specifying creation of the command and registering command. For writing an own command at least one of the already provided command-interfaces has to be implemented.

5. Conclusion

The framework was developed to enable intelligent high level control for mobile robots. In this paper we presented the architecture and mentioned important features of the framework. Of course, the usage of this HLC framework has much more latency than LLC operations, so what level to use depends on the context. In the framework, simple LLC operations could be done by working directly with the CBCJVM classes (e.g. in extra classes). In further steps we will enlarge and optimize the framework, in order to reduce latency and simplify the usage.

Acknowledgement

The authors wish to thank the KISS Institute of Practical Robotics, especially Dr. David Miller, for the support and the Austrian Federal Ministry of Science and Research (BMWF) for their financial support with the programme Sparkling Science.

References

[1] Asl, F. M.; Ulsoy, A. G. & Koren, Y.: “Dynamic Modeling and Stability of the Reconfiguration of Manufacturing Systems”, Technical Report, University of Michigan, 2001.

[2] McDorman, B., Woodruff, B., Joshi, A., Frias, J., 2010. CBCJVM: Applications of the Java Virtual Machine with Robotics, Global Conference on Educational Robotics 2010, Southern Illinois University Edwardsville

[3] Duflou, J.R., G. Seliger, S. Kara, Y. Umeda, A. Ometto, and B. Willems. 2008. Efficiency and feasibility of product disassembly: A case-based study. CIRP Annals - Manufacturing Technology 57, no. 2: 583-600.

[4] Kim, H., Kernbaum, S. & Seliger, G., 2009. Emulation-based control of a disassembly system for LCD monitors. The International Journal of Advanced Manufacturing Technology, 40(3), 383-392.

[5] Kim, Kyoung-Yun, David G. Manley, and Hyungjeong Yang. 2006. Ontology-based assembly design and information sharing for collaborative product development. Computer-Aided Design 38, no. 12 (December): 1233-1250.

[6] Kopacek, P. & Kopacek, B., 2006. Intelligent, flexible disassembly. The International Journal of Advanced Manufacturing Technology, 30(5), 554-560.

[7] Koppensteiner G., Merdan M., Lepuschitz W., Hegny I.: "Hybrid Based Approach for Fault Tolerance in a Multi-Agent System"; IEEE/ASME International Conference on Advanced Intelligent Mechatronics AIM2009, Singapore; 2009.

[8] Lazinica, A. Katalinic, B. „Self-Organizing Multi-Robot Assembly System”, International Symposium on Robotics VOL 36, pages 42, 2005,

[9] Matsumoto, M., 2009. Business frameworks for sustainable society: a case study on reuse industries in Japan. Journal of Cleaner Production, 17(17), 1547-1555.

[10] Merdan M. 2009. Knowledge-based Multi-Agent Architecture Applied in the Assembly Domain. PhD Thesis, Vienna University of Technology. http://www.ub.tuwien.ac.at/diss/AC05040230.pdf.

[11] Merdan M., Vallee M., W Lepuschitz., and Zoitl A., “Monitoring and diagnostics of industrial systems using automation agents,” International Journal of Production Research, vol. 49, no. 5, p. 1497, 2011.

[12] Miller, D.P.; Oelke, M.; Roman, M.J.; Villatoro, J.; Winton, C.N.; , "The CBC: A LINUX-based low-cost mobile robot controller," Robotics and Automation (ICRA), 2010 IEEE International Conference on , vol., no., pp.4633-4638, 3-7 May 2010

[13] Torres, F. et al., 2004. Automatic PC disassembly for component recovery. The International Journal of Advanced Manufacturing Technology, 23(1), 39-46.

[14] Vallee M., Kaindl H., Merdan M., Lepuschitz W., Arnautovic E., and Vrba P., “An automation agent architecture with a reflective world model in manufacturing systems,” in IEEE International Conference on Systems, Man, and Cybernetics (SMC09), San Antonio, Texas, USA., 2009.

[15] Zoitl A., Real-Time Execution for IEC 61499. USA: ISA-o3neidaA, 2009, no. ISBN: 978193439-4274.

[16] Botball Educational Robotics Program, http://www.botball.org, last viewed June 2011

[17] JADE – Java Agent Development Environment, Telecom Italia Labs http://jade.tilab.com/, last viewed June 2011

[18] Jess – The Java Expert System Shell, Ernest Freidman Hill, Sandia National Laboratories http://www.jessrules.com/, last viewed June 2011

[19] Protégé – Ontology Editor, Stanford University, http://protege.stanford.edu/, last viewed June 2011

[20] Sparkling Science, BMWF, http://www.sparklingscience.at/en, last viewed June 2011