# Knowledge Driven Mobile Robots Applied in the Disassembly Domain

Gottfried Koppensteiner, Reinhard Hametner, Rene Paris, Alejandro Moser Passani, and Munir Merdan
Institute of Automation and Control, Vienna University of Technology, Austria
Email: {koppensteiner,hametner,paris,merdan}@acin.tuwien.ac.at, alejandro.moser.passani@gmail.com

***Abstract*—Mobile robots can be used as a motivating and interesting tool to perform laboratory experiments within the context of mechatronics, microelectronics and control. Considering the disassembly as a vital and prospective industry domain, we use the mobile robots to automate the disassembly process. In our system, each mobile robot has particular skills and is supervised by an agent with related objectives and knowledge. An agent has an ontology-based world model, which is responsible to maintain the knowledge about the robot's activities in relation to its environment as well as to its underlying software parts. The ontology is used to represent a specification of an agent's domain knowledge. The system functionality is tested with three mobile robots having a task to disassembly a particular Lego construct.**

## I. 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 [10], PCs [15], 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 [9]. 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 [11]. 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 behavior. 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.

## II. Agent Architecture

To facilitate the design of multi-agent control systems, in a previous work a generic agent architecture [7], [12], [16] 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 [17]. 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. To enable agents to understand the transmitted messages between the LLC and HLC, a representation of the used datatypes as well as the *MessageContent* concept and its sub-concepts is included in the agent's ontology [13]. Fig. 1 shows the architecture of the system which provides the HLC for the mobile robots, with focus on the interplay of the used technologies.

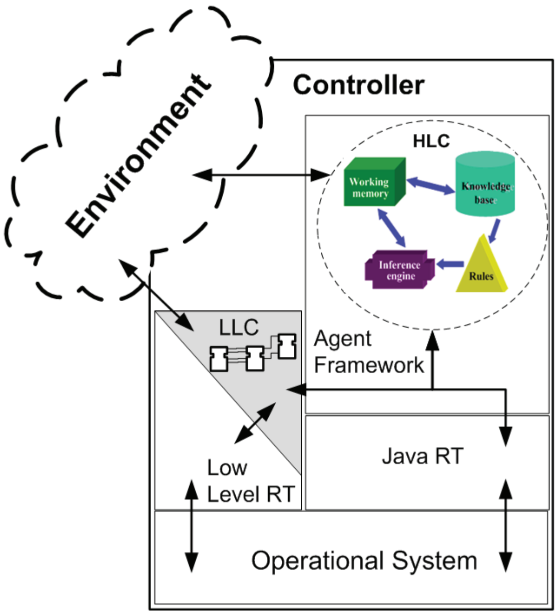

Fig. 1. Agent Architecture includes the HLC and LLC Overview [8]

The whole system 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 behavior 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. The system function is presented in the Fig. 2.

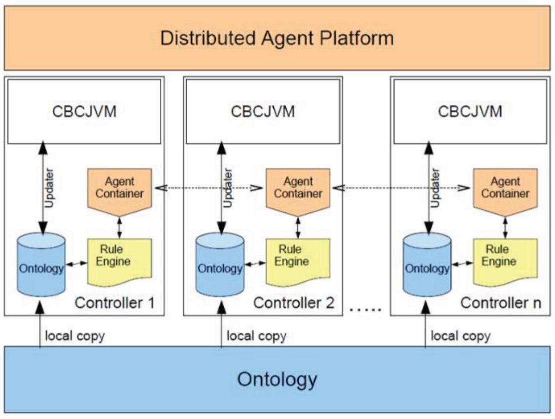

Fig. 2. System Architecture of the HLC

## III. System Integration

### A. A LINUX-based Low-cost Mobile Robot Controller

The basis of the implemented High Level Robot Control is the CBCv2 [14] 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 [18]. Due to the aim of the sparkling science initiative [22], 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.

### B. Framework

On top of the linux-based robot controller, explained in the previous chapter, the CBCJVM is used. CBCJVM uses a lightweight Java Virtual Machine (JVM) also called JamVM designed for embedded platforms which makes it perfect for the use on the CBC. The CBCJVM includes Libraries which allow access to the KISS-C libraries via the Java Native Interface (JNI), allowing to control motors, read sensors, and so on. On top of this, the JADE-Leap platform takes place. JADE [19] simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications [23]. The LEAP libraries allow obtaining a FIPA-compliant agent platform with reduced footprint and compatibility with mobile Java environments. Together this is the host for the Agents, which finally control the robots. Within the agents rule-based-behaviors a knowledgebase is used to map different states of the sensor and motors as well as actual work pieces or other robots. The rule-based behaviors are written in the JESS Language and fit to mechanical possibilities of the robot, enables the Agents to know what it is able to do and which actions it could perform.

### C. Agent Communication

As already mentioned in the system architecture, there's the possibility to share 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*.

## IV. System Implementation

### A. 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. Fig. 3 shows selected classes from the framework's ontology, some of their attributes and their relationships to each other. The class CBC Controller inherits from controller. While the Controller class stores a list of all sensors and actors and is an entry points for future extensions, the CBC Controller adds a more precise view of these, including the software buttons, the acceleration sensor, and the camera.

The following code samples in Fig. 4 will show how rules, implemented in the Java Expert System Shell (Jess) Language [20] can use different sensors in an environment where multiple controllers are present.

Due to memory constraints of the CBCv2, it was not possible to use the Protg [21] ontology API at runtime. Instead, the 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

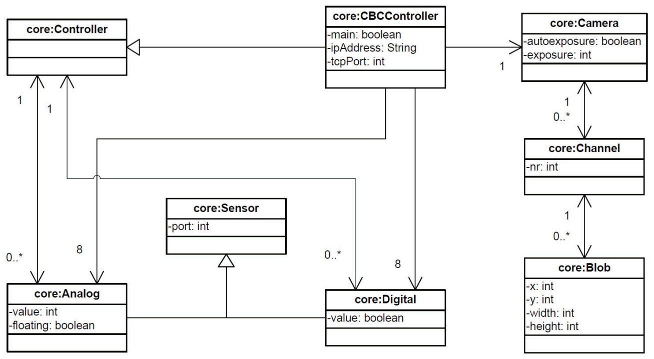

Fig. 3. Part of the ontology about the robot system which includes sensors and the vision system.

```
(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)
)
```

Fig. 4. Jess Rule-based Language - left part: If "A"-Button is pressed on the controller the engine starts; right part: If an analog sensor has a certain value, the engine is switched off.

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.

### B. Interfaces

- Modify Ontology: The ontology 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.
- 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 behavior, where the functionality of the agent is stored. According super classes are *CyclicBehavior* and *OneShotBehavior*.

- 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.

### C. Vision - Object Recognition

To obtain through artificial vision the position of the Duplo bricks it was chosen to use the OpenCV libraries [24] that allow to speed up the task of image processing, as they include more than 500 functions. It was also decided, to divide the task into 3 steps:

- Object detection
- Verification and obtaining of the points that define the object
- Calculation of its position

For the first stage, and after segmenting the image by colors, the function *cvHaarDetectObjects* was used. The *Haar* classifier was trained to find the brick holes, see Fig. 5(a). In case of an 8-holes-brick the second step was to verify that 8 holes were found. For achieving it, it was developed a function that had to search for points placed on the same straight line. If the function found 4 points with a similar distance between them, it was given as correct. If a second line, parallel to the first one was also given as correct, then the brick was considered verified. The last step lied on calculating the bricks position. After a camera calibration and using the *cvFindExtrinsicCameraParams2* function, all the rotation and translation parameters on the three axes were obtained. They allowed to locate each brick with high precision, see Fig. 5(b), and therefore the possibility to adjust the robot in a way to the brick, so that it will be able to hold, lift, or grab the brick.

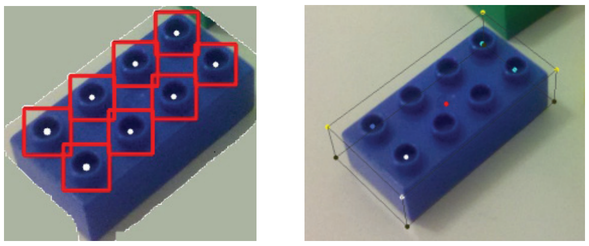
(a) Object Finder (b) Calculating Position

Fig. 5. Object recognition of Lego brick

## V. Test Case

To avoid too many external complications regarding different tool sets and motions when handling goods assembled with screws etc., the focus was only on Lego-Assemblies. This allows building simple robots, even built up on Lego and few metal parts as well as to keep the influencing factors down to a minimum and be able to focus more on the robots intelligence. Therefore, three different robots each equipped with a camera where used to disassembly four bricks of Lego in this test case. There is one robot to grab the assembly (4 Lego bricks stacked together), one robot to lift one brick from the assembly, and one robot sort the different disassembled bricks to the storage. Each robot has knowledge about itself, its possibilities and about allowed Lego-bricks used for the assembly. There are 4 different colors used for the bricks, which should be sorted according to their color after their disassembly.
In a lab environment the robots are doing pretty well. In case of the camera there are some problems with background noise. The collision detection during path planning as well as the mechanics of the robot itself should be enhanced, but the focus of this project was to proof that multi-agent-systems are well situated for the usage in mobile robotics within the disassembly process.

## VI. Conclusion

In this paper we presented the architecture and mentioned important features of the framework. The framework was developed to enable intelligent control for mobile robots. Of course, the usage of this HLC framework has much more latency than LLC operations, so what level to use depends on the context. There is a lot of computation time, but also a high latency caused by the usage of JamVM, JADE and JESS together with camera functions on the Controller.
In further steps we will enlarge and optimize the framework, in order to reduce latency and simplify the usage. At present we are only able to handle few types of Lego bricks assembled in only one way. Our future work will focus more on the object recognition and the mapping of the results into the agents knowledge base, so that the agents are able to reason about the given assembly and possible disassembly steps.

## Acknowledgment

The authors want to thank Clemems Koza, Christoph Krofitsch, Manuel Parg, and Antonio Prvulovic from the Vienna Institute of Technology for their participation within the development of the framework, as well as the internship-pupils-team for building up the simple robots. Furthermore we want to thank the KISS Institute of Practical Robotics, especially Dr. David Miller and Steve Godgame, for the support and the Austrian Federal Ministry of Science and Research (BMWF) for their financial support of the project DISBOTICS at the Sparkling Science program.

## References

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

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

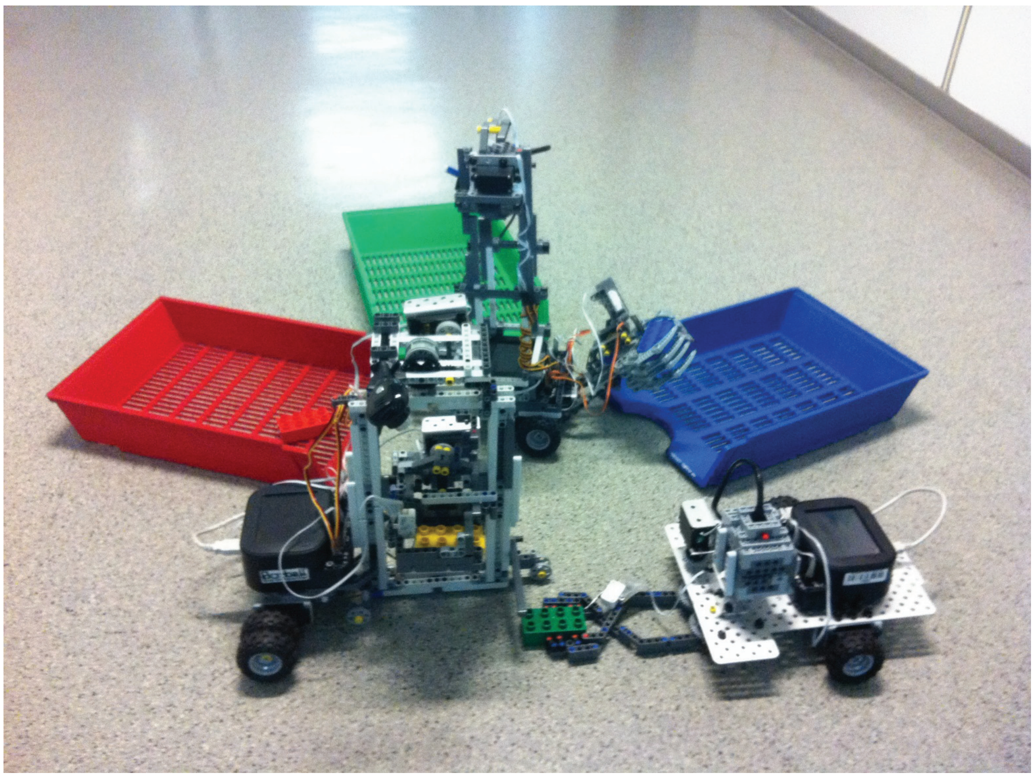

Fig. 6. Test Case with 3 Robots; left robot lift one brick from the other brick, middle robot sort bricks according to their color into the storage area, right robot hold different types of Lego assemblies; all robots are equipped with the camera vision system to find bricks

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

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

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

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

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

[8] G. Koppensteiner, M. Merdan, I. Hegny, W. Lepuschitz, S. Auer, B. Groessing, *Deployment of an ontology-based agent architecture on a controller*, 8th IEEE International Conference on Industrial Informatics, Osaka, Japan, 2010

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

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

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

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

[13] M. Merdan, W. Lepuschitz, I. Hegny, and G. Koppensteiner, *Application of a Communication Interface between Agents and the Low Level Control*, Proceedings of the 4th International Conference on Autonomous Robots and Agents, Wellington, New Zealand, 2009.

[14] D.P. Miller, M. Oelke, M.J. Roman, J. Villatoro, and C.N. Winton, *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

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

[16] M. Vallee, H. Kaindl, M. Merdan, W. Lepuschitz, E. Arnautovic, and P. Vrba, *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.

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

[18] KISS Institute of Practical Robotics, *The Botball Season* http://www.botball.org, Accessed May 2011.

[19] Telecom Italia Labs, *JADE - Java Agent Development Framework*, http://jade.tilab.com/, Accessed March 2011.

[20] Sandia National Laboratories, *Jess: the Rule Engine for the JavaTM Platform.*, Available at: http://herzberg.ca.sandia.gov/, last visited May 2011.

[21] Stanford Medical Informatics, *Protégé Ontology Editor*, Stanford University. Protégé Website. http://protege.stanford.edu., Accessed May 2011

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

[23] The Foundation for Intelligent Physical Agents, *F*IPA Specifications, http://www.fipa.org/specifications/index.html, last viewed July 2011

[24] WillowGarage, *OpenCV (Open Source Computer Vision),* http://opencv.willowgarage.com/wiki/, last viewed July 2011