Decision-Making Embedded Devices using FIPA-ACL Communication

Pascal Proksch, Clemens Westerkamp, Juergen Wuebbelmann
Faculty of Engineering and Computer Science, UAS Osnabrueck
{P.Proksch, C.Westerkamp, J.Wuebbelmann}@fh-osnabrueck.de

***Abstract*- The LK³S approach uses FIPA compliant agent technology to realize a flexible and easy to configure decision-making on embedded devices. The agents are developed by using the LK³S configuration application generating an agent description out of UML diagrams. To enable agents on embedded systems with limited resources an LK³S Embedded Agent Framework (LEAF) is created, which uses the Agent Management System and Domain Facilitator of the hosting agent system in LK³S (JADE).**

I. INTRODUCTION

The objective of the *Easy Configurable Components in Collaborative Systems* (LK³S) [1] project is to build up an environment using agents for heterogeneous systems in automation industry and logistics (e.g. material flow). The advantages of using agent technology are that agents behave proactively and can be decentrally organized. In 2002, the FIPA (Foundation for Intelligent Physical Agents) standardized several aspects of agent communication and management [2]. Because of the objective to be usable on heterogeneous systems only FIPA compliant agent systems are used.

With PABADIS'PROMISE [3] an approach to use agent technology in an industrial environment already exists. PABADIS is built upon the agent system Grasshopper. Agents are used to achieve a higher flexibility, adaptability and speed, when it comes to organization of production and supply-chain management. The approach is focused on controlling and networking of embedded control systems in manufacturing enterprises e.g. at ERP and production level. As written in [4], there has also been an attempt to realize a system wide consistent communication in the FIPA-ACL (FIPA Agent Communication Language [5]). Difficulties with the FIPA-ACL support of Grasshopper prevented a completely consistent communication. So FIPA-ACL was only realized for agent communication, while the actual PABADIS communication was realized by method invocation. Due to the high hardware requirements of Grasshopper, there were also problems using the agent system on devices with limited resources.

The effort of LK³S is to use agents on every participating device, even on devices with limited resources. This leads to a consistent way to describe the behavior and communication of the devices within the LK³S. Non FIPA compliant systems can be attached using Gateways. In Figure 1, the ERP system

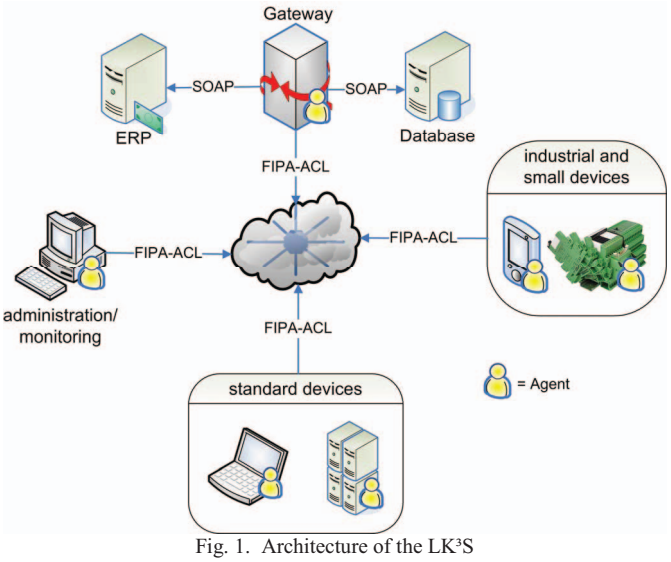

Fig. 1. Architecture of the LK³S

and an external database application is attached to a Gateway using a Web Services interface to communicate with the LK³S environment.

Another project having a consistent approach for system wide communication is SOCRADES ([6], [7]). The difference to LK³S is that instead of agents Web Services are used for the consistent communication and organization. The integration of devices into the web-service world is realized with DPWS (Devices Profile for Web Services). The DPWS specification defines a minimal set of Web Service protocols useful in realizing network architectures of Web Service-enabled devices (e.g. embedded systems).

In SOCRADES agent technology is used to represent non physical DPWS Clients or DPWS Servers for simulation [8]. These agents are used e.g. to simulate a scenario with 5000 temperature devices, while physically only 50 temperature sensors are available. The other 4950 temperature sensors are represented by agents. This makes large-scaled evaluations possible, reducing evaluation costs and complexity.

There were also projects at the Rockwell Automation Research Center [9] to simulate agent based manufacturing control systems with agent technology. The way the controlling agents act were simulated by an emulation subsystem (Matlab, Arena or MAST) and were compared. To increase the accuracy the controlling and simulating agents are able to access the data table of the PLC (Programmable Logic Controller) directly.

978-1-4244-3760-3/09/$25.00 ©

The LK³S is similar to this approach, but does not try to replace full traditional central systems. It is an effort to enrich these systems by replacing parts with agent technology. Additionally, it is an attempt of the LK³S project to keep everything easily configurable. This also includes the control and simulation agent so that it is possible to easily adjust the agent's behavior or decision metrics.

Agents usually work decentrally, while automation industry and logistics are centrally organized using ERP systems, where all information converges. The LK³S approach should not be seen as an attempt to try to replace this central organization. Instead the approach tries to merge both worlds. The effort of LK³S is to improve this purely central approach by adding decentralized decision-making agents, where this approach leads to optimization. This also includes having agents on devices with limited resources (e.g. embedded systems) as mentioned before.

In fact, each agent has to be individually adjusted to the conditions and workplace tasks, which normally requires fundamental knowledge of the agent system itself. To simplify this process, the LK³S project develops a configuration application using a subset of UML diagrams (e.g. activity and state diagrams) to generate agents. UML diagrams are used, because they are easy to create, modify and understand (even for non engineers). Most experts on building up organization or business processes may also be able to control or modify these diagrams, without having the knowledge about the technical realization.

Once the LK³S platform is installed on a device, the agents will be automatically loaded from the agent directory. The device may be used directly by other devices without any manual reconfiguration. This approach is similar to a plug & play behavior.

The capabilities of agents are not only limited to plug & play. If desired, agents can operate in a self-organizing and self-optimizing manner. If an agent announces its capabilities to the agent system, another agent may communicate with this agent making a proposal to use a capability of the agent. When there are multiple agents offering the same capability at the same time, the agent might choose the "best" or "most attractive" offer for this task by using a specific customizable metric (e.g. energy efficiency, time, costs or some combination). It is foreseen that the ERP system is able to provide these metrics. This offers the chance to dynamically adapt metrics to the actual overall strategy.

Additionally, the LK³S supports simulation and benchmarking by using agents to simulate the participating components and devices. The simulation of the LK³S behavior and comparison to existing logistic solutions is realized using event sequences from existing scenarios as stimuli for the agent simulation. The benchmarking results clearly show whether the resources have been used optimally or if optimization potential exists (e.g. structure of storage, if restructuration is necessary). What-if scenarios and simulated deviations can be used to analyze the robustness and efficiency of an agent-based approach. Moreover, all

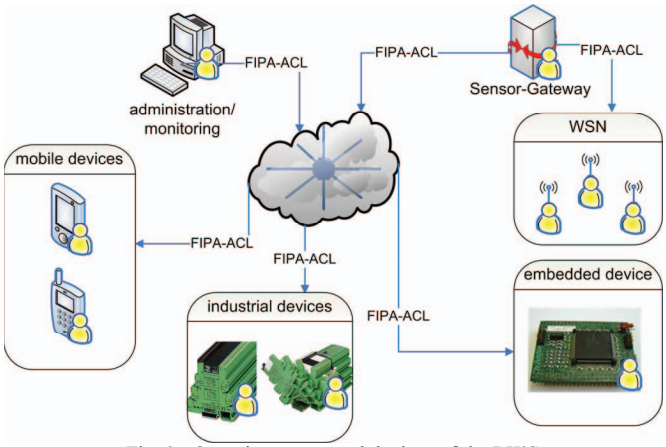

Fig. 2. Overview supported devices of the LK³S

decisions of agents are logged and visualized to allow the monitoring of the minimization of down times and reconfiguration.

All in all, the LK³S is a system that tries to bring the advantages of agent technology into the automation industry.

II. System Wide Communication using FIPA-ACL

As shown in Figure 2 and mentioned above, the communication between the LK³S components is FIPA-ACL based. The LK³S environment uses the FIPA compliant *Java Agent Development Environment* (JADE [10] and [11]) on devices where the Java Runtime Environment is available.

Gateways are not only used to maintain an interface to non agent applications. They can also function as a bridge for network connections to attach devices, where no support for TCP is available.

The approach of LK³S is to provide FIPA compliant agents on embedded systems with limited resources, too. The advantage of using agents on these systems is a seamless integration of the embedded systems into the overall system. They are an integral part of the system having the same interfaces and using the same protocol. Moreover, they are not addressed separately, which would lead to protocol conversions.

There will also be an extension to the UML configuration application, which integrates code generation for the agents of the embedded platform. The advantage of this approach is that all agents are created and modified using only one application, allowing the agent developer to stay in one design domain.

Having the ability to easily design decision-making for embedded systems and changing the behavior just by exchanging the agents on the target leads to a better and more flexible integration of embedded devices in the overall system.

The drawback of this approach is a higher footprint and message overhead. Additionally, the communication between the agents does not fulfill realtime requirements. However, the realtime requirement on the embedded platform is not

violated, as the agents depend on the scheduling of the realtime scheduler of the embedded operation system.

FIPA-ACL messages are designed to contain all data necessary for agent communication. Although, there is a format, which is predestinated for the use with embedded systems with the FIPA-ACL for bit efficient encoding standard ([12] and [13]) the overhead can not be ignored. The requirements on the devices hardware will increase, but the overall advantages of this approach may prevail. These systems with specifically low resources could be easily addressed by using proxy agents.

III. Modeling Decision-Making process using UML Diagrams

As it is an effort of the LK³S project to keep everything easily configurable, there is a configuration application to create and modify all agents out of UML diagrams. The possibilities for designing an agent in UML reach from describing the behavior, sending and receiving messages and the way of decision-making. Important factors for the decision–making could be the local and system wide metrics.

The Figures 3 and 4 show an example of describing an agent with a statechart and activity diagram. There are other approaches like AgentUML [14] describing agent behavior in other UML diagram types, as well. The definitive selection, which UML diagrams are suitable to generate an agent description is not finished, yet. The results of the studies will be published in other publications in the near future.

The following example is designed to have a simple decision-making logic and to show how code generation out of these diagrams can be achieved. The scenario of the example is a logistic application of a LK³S project partner, where we have a storage area with pallets which are due to be inserted into the storage. Each pallet has an representing agent (pallet agent) which is stuck to the pallet during its lifecycle in the storage. Also each forklift is represented in the agent system by its own agent (forklift agent).

The example displays the role and actions of the pallet agent in this agent based storage scenario. The diagram in

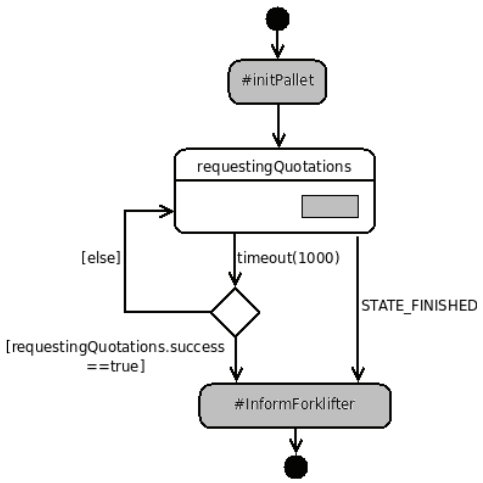

Fig. 3. Toplevel statechart diagram roughly describing an agents lifespan

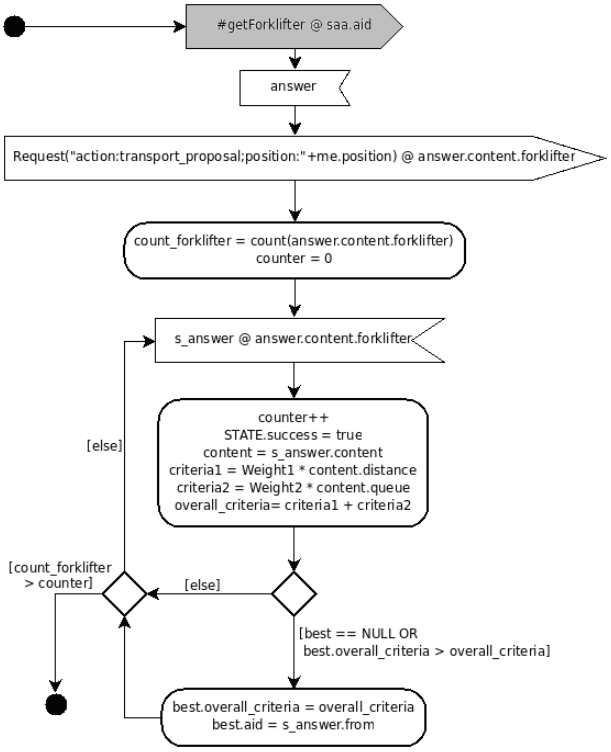

Fig. 4. Detailed Activity diagram describing the requestingQuotations state

Fig. 3 shows that the pallet agent consists out of three states: *initPallet, requestingQuotations* and *informForklifter*.

First the pallet agent gets initialized within the *initPallet*-state, registering at the Domain Facilitator (DF) and loading the actual metrics for the decision-making process. In this example this state is a predefined element. Predefined elements can be imported into the diagram out of the configurator database and it is possible to customize and create them with the configurator. There are many possibilities for storing and loading UML-elements in the configurator database, but these aspects are not part of this article.

Afterwards, the *requestingQuotations* state is entered, which is displayed in Fig. 4. This state is also triggered by a timeout which interrupts the internal state processing if the execution time exceeds the timeout value (in this example 1000ms)

In the *requestingQuotations* state, the agent sends and receives various messages. Additionally, the agent calculates and compares metrics for his decision-making. To simplify the diagram it is possible to use predefined activities and messages out of the configurator database.

Overall, the agent sends a predefined message to the saa (Storage Area Agent) requesting a list of all available forklift agents. Afterwards, the agent waits until it receives a message and stores it into the variable “answer”. Then the agent sends a *FIPA-ACL Request* to all forklift agents on that list. Finally, the agent processes each receiving message until all forklift agents have answered or the timeout interrupts the execution. After each received message the pallet agent compares the

actual offer with the currently best received offer. If the actual offer is better, this offer becomes the best received offer.

As shown in Fig. 3, if the state is ended properly the agent moves to the *informForklifter*-state. If the timeout occurred, a case differentiation is necessary. If the state was successful (at least one answer received), the *informForklifter* state is reached, and otherwise the *requestingQuotations* state gets repeated.

In the *informForklifter* state the agent informs the forklift agent, with the best proposed offer. The forklift agent appends the pallet into its working queue.

IV. Configurable Embedded C-Agent Development Environment

A. Requirements for Agents on Embedded Systems

The requirements of an agent system for a seamless integration of embedded systems in the LK³S project are relatively high. First of all, the agent system has to support FIPA-ACL communication to be able to communicate directly with JADE agents. Secondly, the resource consumption has to be so low that it can be used even on devices with limited resources. Additionally, most embedded systems come up with real-time requirements, which can not be addressed by the agents itself. There also has to be a possibility for the agents to monitor and control these processes. For a model driven approach a defined interface is used.

B. Existing approaches for agents on embedded systems

There are some existing approaches to use FIPA compliant agents on systems with low resources. Nearly all approaches are written in Java (e.g. Jade Leap [15], Aglets [16] and Voyager [17]) and require at least a Java Micro Edition Runtime (e.g. CLDC and MIDP) on the embedded system. In [18] the memory requirements of the CLDC Java Environment are mentioned to be between 160 and 192 KB. Additionally, the JADE LEAP Classes require about 100 KB so that the memory consumption of JADE Leap is also acceptable for systems with low resources. The problem with using JADE LEAP is that on most embedded platforms Java is not available. To use JADE a Java Runtime would have to be implemented, which would rapidly increase porting complexity on small heterogeneous platforms. Additionally, most applications executed on embedded systems are written in C so the interoperation between the Java based agent system and the C based application could lead to difficulties and performance issues.

Another approach is to use C based agents. This can be achieved using a C/C++ interpreter for developing FIPA compliant agents, e.g. Mobile C [19]. Mobile C is built upon the proprietary *Embedded Ch* interpreter which makes it impossible to use Mobile C with devices and platforms not supported by the interpreter. The available hardware and memory of the supported embedded platforms, specified at the project homepage exceeds the possibilities of most of our embedded systems by far (e.g. 16 MB RAM and more).

C. The Embedded Agent Container Architecture

Although there are many existing agent systems for systems with low resources no system fulfills the requirements for a seamless integration into LK³S environment completely.

To address this shortcoming LK³S develops a FIPA compliant agent framework for embedded systems, which is application and platform independent. It is written in ANSI C, which is supported in almost every embedded system having the GNU-GCC or a proprietary compiler. The *LK³S Embedded Agent Framework* (LEAF) can provide one or more Embedded Agents on a device, which are defined in an interpretable macro language. Important elements of this language are e.g. retrieving data from various internal and external sources, sensor data merging, calculating optimization metrics or generic decision making approaches and parsing or assemble FIPA-ACL Messages. LEAF is attached to JADE, which provides the Agent Management System (AMS) and the Domain Facilitator (DF). The Embedded Agents are able to register at the DF the same way as JADE Agents do using FIPA-ACL communication. By using the JADE AMS and DF the Embedded Agents integrate seamlessly into the JADE Framework, and therefore into the LK³S Framework.

As shown in Fig. 5 the LEAF architecture is divided into different components. The *Embedded Agent Container* (EAC) handles the agent communication fulfilling the FIPA-ACL for bit efficient encoding standards, which is also supported by JADE. Additionally the EAC also provides an interface to the *Embedded Application Objects* (EAO) which are used to abstract and integrate device or application dependent functions. These functions often have real time requirements or need direct access to the *Hardware Abstraction Layer* (HAL) or to the *Operation System* (OS). This reduces the complexity of the agent e.g. an agent does not need to know how to read out a sensor in detail, it is sufficient that the agent knows that the device has the ability to read out the sensor

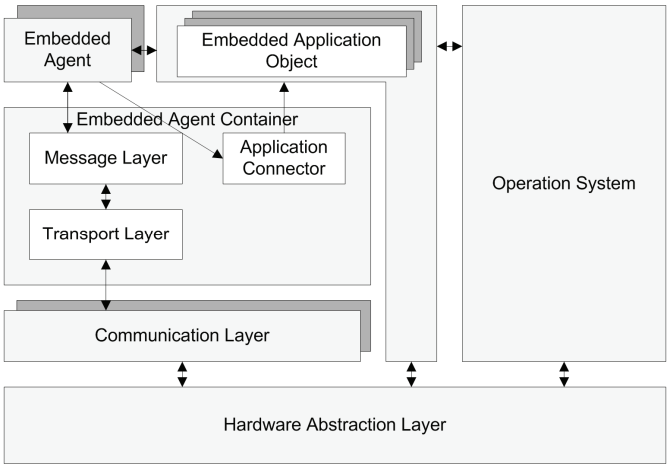

Fig. 5. Architecture of the LK³S Embedded Agent Framework (LEAF)

and which EAO handles this functionality. Each EAO has a system wide unique identification and can be offered on various embedded devices. The source code of the embedded application objects varies from device to device but the underlying functionality remains the same (e.g. reading out a temperature sensor, switching on an LED).

Not every embedded system has access to an ethernet interface. Some may only have serial interfaces or have wireless communication interfaces like Bluetooth, ZigBee or WLAN. Some systems may also have multiple interfaces. Each communication interface is represented to the EAC by an implementation of the *Communication Layer* which provides a generic interface to the EAC (for sending messages to the HAL) and the HAL (to deliver incoming messages to the EAC).

D. Generating Embedded Agents

As already mentioned above, the agents are defined in an interpretable macro language. This language has a reduced instruction set, which only contains some simple logical operations, operations for assembling and parsing the content of FIPA-ACL messages and operations to interact with functions of EAOs. The agent description contains two parts: the code part, where the instructions are placed and the data part, where local variables and constants are stored. Each Agent has a small state machine consisting of an instruction pointer and four registers to store values.

For the application developer the internal embedded agent realization is not visible or important. The most common and easiest way to create or modify an embedded agent description is to use the LK³S configuration application to generate the interpretable code out of an UML (e.g. activity or state diagram). The difference between generating Embedded Agents and JADE Agent generation is, that the functionality and size of the diagram is reduced. Additionally, the EAOs can be used within the activity diagram to be assigned to an Embedded Agent.

E. Distributing Embedded Agents

Every EAC contains a startup agent, which handles initialization on startup. This agent builds up a connection to the DF and searches for an agent with the ability to configure the embedded device (usually a JADE agent) by sending agent descriptions to the startup agent. The startup agent sends a request to the local EAC and creates an agent out of the description. This is shown in Fig. 6. Finally, when all agents are created the startup agent gets disabled and all other agents will be started. Currently, when modifying an Embedded Agents description all devices have to be restarted to apply the changes. It is planned to replace agents without restarting the embedded system.

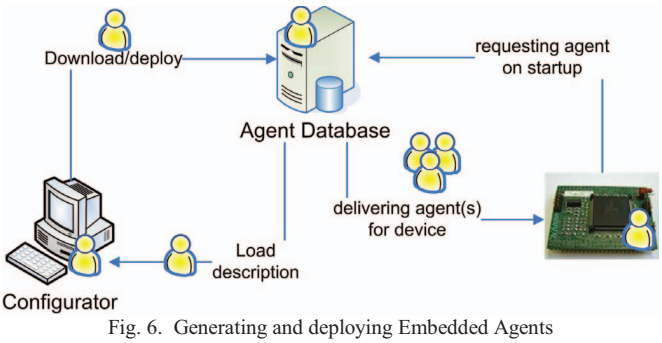

Fig. 6. Generating and deploying Embedded Agents

V. Conclusion

Decision-making embedded devices in LK³S are realized by using agent technology. Therefore a FIPA compliant agent framework for embedded devices (LEAF) is developed and connected to the existing JADE agent framework.

Although LEAF is written for embedded devices, it can also be used on other systems. This may come handy when it comes to bridge network interfaces (e.g. serial-to-ethernet communication) when an embedded device is attached. Additionally, it is an alternative for systems where a Java Runtime Environment is not installed.

By the combination of JADE and LEAF the ontology of JADE will be used for the communication to the AMS and DF. All LK³S specific communication will use an LK³S ontology, which is designed to be parsed more efficiently on embedded platforms.

Due to the FIPA compliance of LEAF it can also be used in other projects using agent technology. This might be useful especially if embedded devices must be integrated, too. Further research projects at the UAS Osnabrueck will use the LK³S platform and extend the coverage of target devices (e.g. mobile devices, visualization terminals, WSN devices, PLCs).

Acknowledgment

This work was supported by the Federal Ministry of Education and Research (BMBF), the industrial partners Phoenix Contact GmbH & Co. KG, Willert Software Tools GmbH, BeKa Engineering, Schneider Electric and SENGATEC and the UAS Emden, which are working on the real time aspects of the LK³S approach.

References

[1] U. Schmidtmann, et al., „Leicht Konfigurierbare Komponenten Kollaborativer Systeme (LK³S),“ *6th Automatisierungstage 2008, on,* pp. 192 – 198, Jan. 2008.

[2] Foundation for Intelligent Physical Agents, “Abstract Architecture Specification.” 2002.

[3] J. Peschke, A. Luder, and H. Kuhnle, “The pabadis’promise architecture - a new approach for flexible manufacturing systems,” *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on*, vol. 1, pp. 6 pp.–496, Sept. 2005.

[4] Q. Feng and G. Lu, “Fipa-acl based agent communications in plant automation,” *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA ’03. IEEE Conference, vol. 2*, pp. 74–78, vol. 2, Sept. 2003.

[5] Foundation for Intelligent Physical Agents, “FIPA ACL Message Structure Specification.” 2002.

[6] A. Cannata, M. Gerosa, and M. Taisch, “Socrades: A framework for developing intelligent systems in manufacturing,” *Industrial Engineering and Engineering Management, 2008. IEEM 2008. IEEE International Conference on*, pp. 1904–1908, Dec. 2008.

[7] F. Jammes, A. Mensch, and H. Smit, “Service-oriented device communications using the devices profile for web services,” in *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing.* ACM New York, NY, USA, 2005, pp. 1–8.

[8] S. Karnouskos and M. Tariq, “An Agent-Based Simulation of SOA-Ready Devices,” in *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*, 2008, pp. 330–335.
[9] Vrba, P.; Marik, V., "Simulation in agent-based manufacturing control systems," *Systems, Man and Cybernetics, 2005 IEEE International Conference on* , vol.2, no., pp. 1718-1723 Vol. 2, 10-12 Oct. 2005
[10] F. Bellifemine, G. Caire, D. Greenwood, and I. NetLibrary, *Developing Multi-agent Systems with JADE*. Springer, 2007.
[11] F. Bellifemine, A. Poggi, and G. Rimassa, “JADE–A FIPA-compliant agent framework,” in *Proceedings of PAAM*, vol. 99, 1999, pp. 97–108.
[12] Foundation for Intelligent Physical Agents, “ACL Message Representation in Bit-Efficient Encoding Specification.” 2002.
[13] Foundation for Intelligent Physical Agents, “Message Transport Envelope in Bit-Efficient Encoding Specification.” 2002.
[14] J. Odell, H. Van Dyke Parunak, and B. Bauer, “Representing Agent Interaction Protocols in UML,” *ICSE 2000 Workshop on Agent-Oriented Software Engineering* (AOSE-2000), June 10, 2000, Limerick, Ireland.
[15] A. Moreno, A. Valls, and A. Viejo, *Using JADE-LEAP implement agents in mobile devices*. Universitat Rovira i Virgili. Departament d’Enginyeria Informatica, 2003.
[16] D. Lange and M. Oshima, “Mobile agents with Java: The Aglet API,” *World Wide Web*, vol. 1, no. 3, pp. 111–121, 1998.
[17] G. Glass, “ObjectSpace Voyager-The Agent ORB for Java,” *LECTURE NOTES IN COMPUTER SCIENCE*, pp. 38–55, 1998.
[18] J. Lawrence, “LEAP into Ad-hoc Networks”. *Workshop on Ubiquitous Agents, Bologna, 2002*.
[19] B. Chen, H. Cheng, and J. Palen, “Mobile-C: a mobile agent platform for mobile C/C++ agents,” *SOFTWARE PRACTICE AND EXPERIENCE*, vol. 36, no. 15, p. 1711, 2006.