SIMULATION IN AGENT-BASED CONTROL SYSTEMS: MAST CASE STUDY

Pavel Vrba[1], Vladimír Mařík[1,2]

[1]Rockwell Automation Research Center, Pekařská 10a, 155 00 Prague, Czech Republic
{pvrba, vmarik}@ra.rockwell.com
[2]Gerstner Laboratory, Department of Cybernetics, Czech Technical University,
Technická 2, 166 27 Prague 6, marik@labe.felk.cvut.cz

Abstract: The paper discusses the role of simulation in agent-based control systems. We present the universal runtime interface enabling efficient interactions between the agent control, manufacturing environment (emulated or physically connected), visualization and low-level real-time control programs. As a case study, we provide the description of the MAST simulation tool and its application to Cambridge packing cell.

Keywords: Agile manufacturing, Distributed Control, Fault tolerance, Simulation, Logic controllers, sensors, actuators, transportation control

1. INTRODUCTION

The term simulation in agent-based control systems refers to much more complex processes than in the case of strictly hierarchical, synchronized control systems. The main difference is that the overall behavior of an agent system emerges from dynamically changing patterns of inter-agent interactions and asynchronously executed decision-making processes of particular agents which are not influenced by any central control element. The emergent behavior, sometimes called aggregate behavior, is expected to form one of the major benefits of this approach (Parunak *et al.*, 1997); however, it can also surface in undesirable ways, thus requiring the attention of the system designers. The integrated cooperation of more diverse modules is usually required as there are many quite specific requirements and expectations put on simulation of the agent-based systems:

a) First of all, we expect that the **qualitative evaluation of emergent behavior** of an agent-based system will be provided. The multi-agent systems have no central element to take the overall responsibility for the behavior of the global community of individual agents which are acting and making decisions locally. Quite new, non-expected – and maybe non-stable – patterns of behavior can appear as a result of interactions of locally operating units. Simulation is expected to detect and analyze suspicious behavioral patterns.

b) Agent-based systems have – in comparison to strictly hierarchically controlled and centralized systems – several specific behavioral features resulting from the principles of agency, namely the ability to react dynamically and appropriately to any structural/functional changes within the agent-based system (an addition or a loss of any agent, decrease of reliability or increase of an overload of an agent, etc.) by means of message passing among the agents as well as by coordination/cooperation techniques (negotiation scenarios, auctioning, etc.). The agent-based

systems are simply able to continue in their operation towards the given goal even in the case that serious changes in the agents' community structure or in the agents' functional capabilities do appear. In consequence of this, **an agent-based system can be simulated only by another agent-based system** – the centralized approach (used e.g. in the existing simulation tools like Matlab or Arena) is absolutely not applicable.

c) The simulation of both the controlled process and the agent-control system has to be provided. Because of the potential direct reusability of the agent-control algorithms, the agent-control part is, as a matter of fact, emulated as well. In such a way **the agent-based simulation is usually organized as an interaction of two emulations.**

d) The main "output" of the simulation processes is the "movie" showing the behavior of the system. The **visualization** subsystem represents an important component of the agent-based simulation package/solution as the visual insight is of key importance for the human-machine interactions in both the periods of the system design and the system operation.

e) The simulation should **run in real time** and be equipped by a **run-time interface** to the real-life control hardware to enable the shift of the borderline between the simulation and real-time real-life control.

f) There are **two kinds of run-time interfaces needed**, namely (i) and interface from the agent control module to the visualization module, and (ii) the machine-machine runtime interface between the agent-control system and the controlled process/manufacturing equipment (either emulated or physically connected – see below).

g) The switching-over of the machine-machine interface between the emulated and physically connected part of the manufacturing equipment should proceed in a very smooth way to enable a **step-wise sequential shifting of the control from the emulated part to the physical part of the equipment**. This would support an efficient commissioning/tuning of the manufacturing systems. In an ideal case, both the interfaces enabling to control the emulated part and the real physical part are identical – this would enable to **directly re-use the agents originally developed for simulation** purposes **for the physical control** of physical manufacturing facility.

2. ROCKWELL AUTOMATION APPROACH

The requirements on simulation tools or platforms for agent-based solutions call for new types of simulation systems (simulation platforms) with embedded principles of agency. One of pioneering systems of this kind, presented in this paper, is the MAST simulation tool being developed by Rockwell Automation (Vrba, 2003).

Another example of such a system is the agent-based control and simulation of the scaled-down form of chilled water system for the US-Navy ships (Maturana, *at al.*, 2004).

The efforts to unify Rockwell Automation agent-based technology lead to the development of a **universal run-time interface** that enables efficient interactions among different components by sharing data (mainly sensor and actuator values) in the data table of the industrial PLC controller. These components include: (i) the physical manufacturing environment being connected through the I/O modules of the industrial controller to transfer the sensor/actuator values; (ii) the subsystem that emulates the behavior of the physical manufacturing equipment; (iii) the agent control system which is responsible for carrying out the agent-based control algorithms for both the simulation purposes as well as for the real-life control of the physical equipment; (iv) the visualization module providing a graphical insight into the manufacturing process state (either the emulated or the real one) as well as into the properties and actions of the agent control system; (v) the low-level real-time control programs (usually in ladder logic) and potentially (vi) other related modules.

The so called data table of an industrial PLC controller is used as a common data memory for holding the sensor and actuator information related to the controlled process as well as for storing other control-related data used by the low-level control programs. The values in the data table are usually referred to as the *tags*.

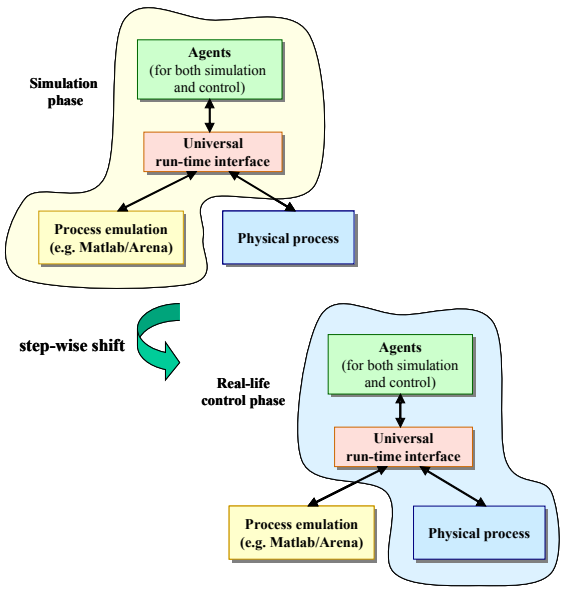

Fig. 1. Universal runtime interface for agent-based simulation and control systems

The idea of the presented universal runtime interface is to enable also other mentioned components (mainly the control agents and the emulation subsystem) to access the data table, i.e. to provide them with the ability to **read** and **write** the tag values.

There are two high-level programming languages used in Rockwell Automation for the development of the control agents: C++ and, more and more preferred JAVA. To allow the agents programmed in these languages to access the data table of the Rockwell Automation's ControlLogix/SoftLogix PLC controllers, we have implemented both the C++ and JAVA API (Application Program Interface) through which the agents (residing either directly within the controller or on a remote PC) can read and write the PLC's tags. Additionally, **create** and **delete** functionality is also supported by the interface to allow to dynamically add or remove tags in the data-table reflecting the structural changes in the control system. For instance, when a new sensor is added at runtime to the conveyor-based transportation system, a set of new elements are dynamically created and added to corresponding subsystems sharing the data-table: the sensor agent is added to the agent control part, the sensor emulation unit is added to the emulating subsystem and the sensor visualization element is added to the visualization module. Concurrently, the tag values corresponding to the state of the sensor are added to the data-table to be shared by these new elements.

As shown in Fig. 1, such an interface is used for both the simulation and real-life control phase. Within the simulation phase when the agent control system is being tested, the manufacturing process/equipment is emulated by the emulation subsystem. It is encouraged to use the commercial simulators like Matlab, Arena, AnyLogic, SolidWorks, etc. for these purposes. In such a case however, it is necessary to implement a link between the simulator and the PLC. (Maturana *et al*., 2004) for instance implemented an OPC connection between Matlab/Simulink and ControlLogix PLC allowing to share sensor/actuator data between the agent-based control and the Matlab emulation of the HVAC (Heating/Ventilation/Air Condition) operations of the US Navy ship.

When switching from the simulation to the real-life control phase, the emulation subsystem is replaced with the physical manufacturing system. A great advantage is that there is no need to do any change to the agent control system since the same runtime interface is used also in this real-life control phase. Additionally, the universal runtime interface allows that a part of the controlled system is still emulated while another part of the system is already physically connected. The shift of the control functionalities from the agent-based simulation towards the physical control equipment is usually carried out in a step-wise way, sequentially. Thus, a smooth transition is achieved. The same applies also to the visualization – if it handles the information from the data table, the same tool can be used for the simulation as well as for the real-life control (in the case of Rockwell solution the RSView software or the PanelView operator terminals can be used).

3. MAST – MANUFACTURING AGENT SIMULATION TOOL

3.1 Brief Description

The Manufacturing Agent Simulation Tool (MAST) represents a new generation of simulation systems with embedded multi-agent systems principles aimed at the manufacturing domain. Initially, the MAST tool has been intended as an agent-based demo application that would illustrate – on some typical manufacturing task – the major benefits of the deployment of the agent technology. As a suitable domain we selected the material handling, particularly the transportation of (semi-)products or discrete materials on the factory's shop floor using a network of conveyor belts or/and AGVs (Automated Guided Vehicles).

We identified the basic types of material handling components such as a work cell, conveyor belt, diverter, AGV etc. and developed the appropriate agent classes that represent these components. The behavior of agents is oriented toward the mutual cooperation in the optimal transportation of products between the work cells that are interconnected via a network of conveyor belts, diverters and intersections. The cooperation is based on sending the short, informative messages between the neighboring agents about the configurations of possible transportation paths in order to select the optimal, i.e. shortest routes between the work cells. It is important to say that there is no central control element – the decision making processes are distributed over the agents that work autonomously without being affected by any central, higher-level element.

To show the robustness and flexibility of the agent solution, the attention is paid to the failure detection and recovery – a failure of any component can be emulated (e.g. a failure of the conveyor belt) that causes the agents to start the negotiations on the alternative transportation paths while avoiding the broken component. An easy-to-understand visualization helps the user to observe the overall behavior of the system during the simulation.

MAST is entirely implemented in JAVA language and uses the JADE agent platform (JADE, 2004) as the runtime environment for the agents. From its first prototype developed more than three years ago, the MAST tool grew up into a comprehensive agent-based control and simulation system consisting of following parts:

1. **The agent control part** that contains a library of JAVA/JADE classes representing the material-handling components (see Section 3.2 for more

details). Basically, each class accounts for the description of the agent's attributes and the agent's behavior. For example, the conveyor belt agent class includes these attributes: name of the agent, names of the neighboring components (input and output component) and cost of the delivery (see bellow). The behavior is specified as a general set of rules according to which all the agents created as instances of particular class will behave. In the case of the conveyor one of such a rule is: "*if I* receive a message from *my* output component informing me about its failure *then I* stop the conveyor *and* forward the message about failure to *my* input component".

2. **The emulation part** that provides the agents with the emulation of the physical manufacturing environment. We have decided not to use any of the existing tools like Matlab or Arena; instead we have developed our own ad-hoc emulator comprised of:
   a) The emulation model of the material handling system consisting of the JAVA objects representing particular components with their virtual sensors and actuators. These emulation objects are appropriately linked with each other according to the structure of the system. For instance the conveyor belt B_2 in Fig. 2 links the diverter D_1 and the intersection I_1. Such a structure along with the specification of attributes of particular components (e.g. the cost of the conveyor B_2) is stored in an XML file.
   b) The emulated products (we refer to them as *work pieces*) being currently transported. Each work piece contains a link to the emulation component by which it is currently being transported.
   c) The emulation engine that moves the virtual products throughout the system. When the emulated product is about to leave a component which is currently on and enter a successive component (e.g. leave conveyor B_1 and enter diverter D_1 in Fig. 2), the sensors of these components (output sensor of B_1 and input sensor of D_1) are called by the emulation engine. This causes to inform the appropriate agents through the runtime interface.
3. **The runtime interface** implemented as direct link (JAVA method calls) from each emulation object to the appropriate agent (sensor signals) and vice versa from the agent to the emulation object (actuator signals). Currently, there is an ongoing work on modifying the runtime interface of MAST in terms of splitting the tight link between the emulation objects and agents. Instead, we will use the above mentioned JAVA API that allows any JAVA application to directly access the data table (memory) of the ControlLogix PLC. Thus, the sensor/actuator values will be shared in the data table and read/written by the emulation as well as agents. As already mentioned, such an interface will allow to replace the emulation part of the MAST tool with the physical hardware and thus to use the agent control part without any additional modifications for the physical manufacturing control.
4. **The GUI** (see screenshot in Fig. 2) for the graphical drag-and-drop design of the material handling system as well as for the visualization of the simulation. Through the GUI the user can send work pieces between the work cells, introduce failures of different components and even change the structure of the system at

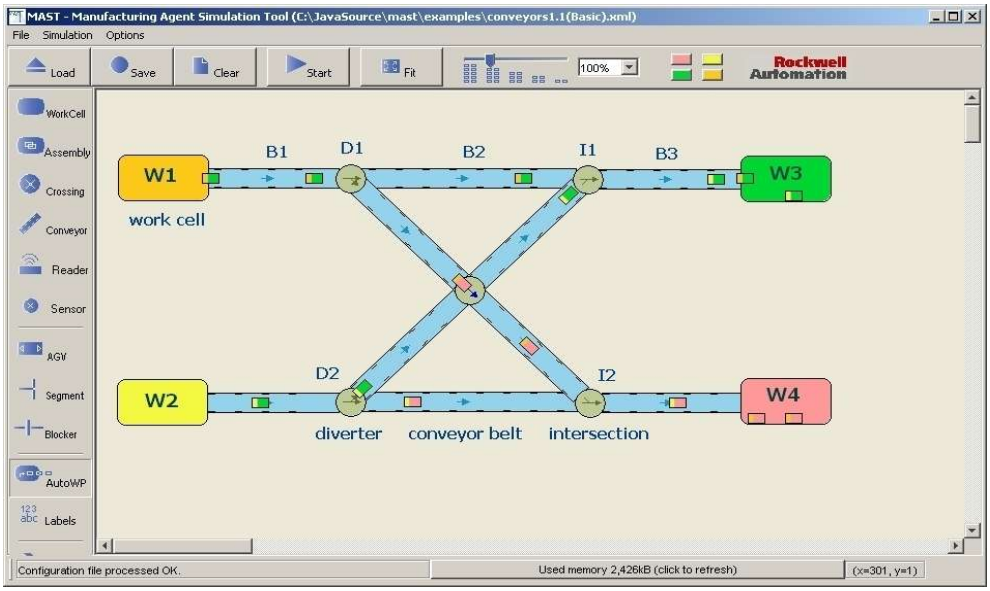

Fig. 2. Screenshot of the MAST tool (conveyor-based transportation)

runtime – any component can be removed (or disconnected from its neighbors) or a new component can be added without the need to stop the simulation (see following sections for how these situations are handled by the agents). As said above, the configuration of the material handling system can be saved/loaded in GUI to/from an XML file.

Each component is represented by the XML tag of the following structure (this example shows the conveyor belt component):

```
<component type="ConveyorBelt" name="B3">
    <additionalAttributes … >
</component>
```

While loading the XML file, we take the advantage of the dynamic class loading feature of JAVA – according to the `type` attribute (`ConveyorBelt` in this example) the MAST system tries to dynamically load the JAVA class of the corresponding name. If such a class is found, its `createInstance()` method is called – this method is responsible for parsing the XML description and accordingly creating the instance of the emulation component as well as the appropriate agent. Such a mechanism easily allows future introduction of new components/agents into the MAST's library without a need of any modifications to the program code of MAST.

3.2 The Basic Set of Agents For Conveyor-Based Transportation

In this section we describe in more detail the agent-control part of the MAST tool, mainly what basic types of agents have been developed and what are the communication scenarios used for the mutual cooperation.

The **work cell** agent (e.g. W_1 in Fig. 2) represents a general manufacturing cell, e.g. a drilling/milling machine, storage area, assembly machine, docking station, etc. It is supposed, that work pieces can enter to the work cell from one of its input conveyors and – e.g. after some processing or on a request from the other agents or the user – can be sent out via the output conveyor to another work cell. It is apparent that work cells play roles of *source* and *destination* components in material handling system between them the work pieces are transported. An example of the transportation task can be a delivery of a particular work piece from W_1 (source) to W_4 (destination) in the system in Fig. 2. As can be seen, the color of the work piece in MAST's GUI corresponds to the color of the destination work cell (in this example green work pieces are transported to W_3 and red ones to W_4); that helps the user to easily observe how the work pieces are being routed throughout the system.

The **conveyor belt** is used to transport work pieces between two other components which it is connected to. Obviously, the additional attributes in the XML description of conveyor include the names of the input and output component and the `cost` attribute expressing e.g. a time-period needed to pass through the whole conveyor or e.g. some money value for the use of the conveyor. The cost-based model is applied for routing the work pieces through the network of conveyors – such a sequence of conveyors leading from a given source work cell to the desired destination work cell is used for which the total cost of the transportation is minimal (total cost is obviously the sum of the costs of particular conveyors). To notify the neighboring components, that there is a connection between them provided by the conveyor, the conveyor agent sends an informing message to both of them after being instantiated. The message, following FIPA specifications (FIPA, 2004), sent for example by the conveyor B_3 to the work cell W_3 (see Fig. 2) e.g. looks like:

```
(inform
   :sender B3
   :receiver W3
   :content (
      <connected as="inConveyor" />
   )
   :language XML
   :ontology material-handling
)
```

The `inform` string indicates the type of the FIPA message and the `sender` and `receiver` attributes denote the names of the sender and the receiver of the message. The `content` attribute represents the content of the message – here the B_3 agent informs the agent W_3, that it has connected to it as its input conveyor (`inConveyor`); similar message is sent to I_1 agent to which the B3 conveyor connects as its output conveyor (`outConveyor` is used instead).

The **diverter** agent (e.g. D_1 in Fig. 2) is responsible for the least-cost product routing – it switches work pieces coming from its input conveyor(s) to such of its output conveyors so that the work piece will reach its destination following the least-cost route. For these purposes the diverter agent holds an up-to-date routing table that contains the names of all the reachable work cells and costs of the transportation at which they can be reached using diverter's particular output conveyors.

The routing tables are determined by mutual cooperation of diverters, conveyors and work cells that exchange knowledge about reachable destinations along with costs of the delivery in a back-propagation manner (i.e. from destinations to sources). This dynamic algorithm is triggered when the `<connected as="inConveyor">` message mentioned in previous is sent by some conveyor to a work cell. The work cell agent, that is now aware of being a potential destination since it has the input conveyor, replays with a message including itself as a reachable destination with zero cost. The conveyor adds its cost and forwards this message to its input (beginning) component. If this is a diverter, it updates

its routing table by merging this knowledge with the reachable destinations lists received from other output conveyors. The resulted routing table (including names of all available destinations and costs by which they can be reached) is then propagated back by the diverter to all of its input conveyors. They add their costs to all destinations in the list and propagate it back again in the same way. Thus the global information about the reachable work cells is distributed locally in the diverters – each diverter knows in which direction to send the work piece (through its particular output conveyor) so that the desired destination is reached following the least-cost route. However, the diverter is not aware of what other possible conveyors in that least-cost route will be used after the work piece has left the diverter's output conveyor – it is up to the next diverters in that route to navigate the work piece properly using the same routing mechanism. The source work cells are in this way also informed which destinations and at which costs are reachable via their output conveyors.

The major benefit of this concept is that it provides an easy mechanism for handling the failures. When the user introduces a failure e.g. to the conveyor belt, the conveyor agent sends a `<failure/>` message to its input component. This message is treated as it was a message containing an empty list of reachable destinations (it is obvious that no destinations can be reached through the failed component). So if this message is received by the diverter agent, its routing table is appropriately updated (by removing the unreachable destinations) and forwarded to all diverter's input conveyors. In such a way the information that some destinations became unreachable or are reachable at different (higher) cost now is distributed throughout the whole system.

3.3 Extension of MAST to Cambridge Packing Cell

Recently, the MAST tool has been extended to be able to simulate the holonic packing cell of the Center for Distributed Automation and Control (CDAC) at the Cambridge University's Institute for Manufacturing. This lab provides a physical testbed for experiments with the agile and intelligent manufacturing focusing particularly on the Automatic Identification (Auto-ID/RFID) systems (Fletcher *et al.*, 2003). This emerging standard for automatic product tracking introduces the Electronic Product Code (EPC) as an alternative to the bar code label. The unique EPC number (e.g. 96 bits code) is embedded in an RFID (Radio Frequency IDentification) tag comprised of small silicon chip and antenna. Reading and writing is done wirelessly via specialized devices – RFID readers – using high or ultra high frequency radio waves.

In the Cambridge packing cell, which actual photo is shown in Fig. 3, the RFID technology is used in controlling the packing of Gillette gift boxes (labeled by number 9 in Fig. 3). The user can select from two types of boxes that can be filled by any combination of three out of four Gillette grooming items (gel, razor, deodorant and shaving foam). The lab physically consists of the following components (numbering corresponds with labels in Fig. 3):

1. Conveyor loops (Montech track) to transport the shuttles that carry boxes. There is one main feeding loop (labeled by number 1 in Fig. 3) and two subsidiary loops leading to robots.
2. Gates (diverters) that navigate the shuttles out of the main loop to the subsidiary loops and vice versa.

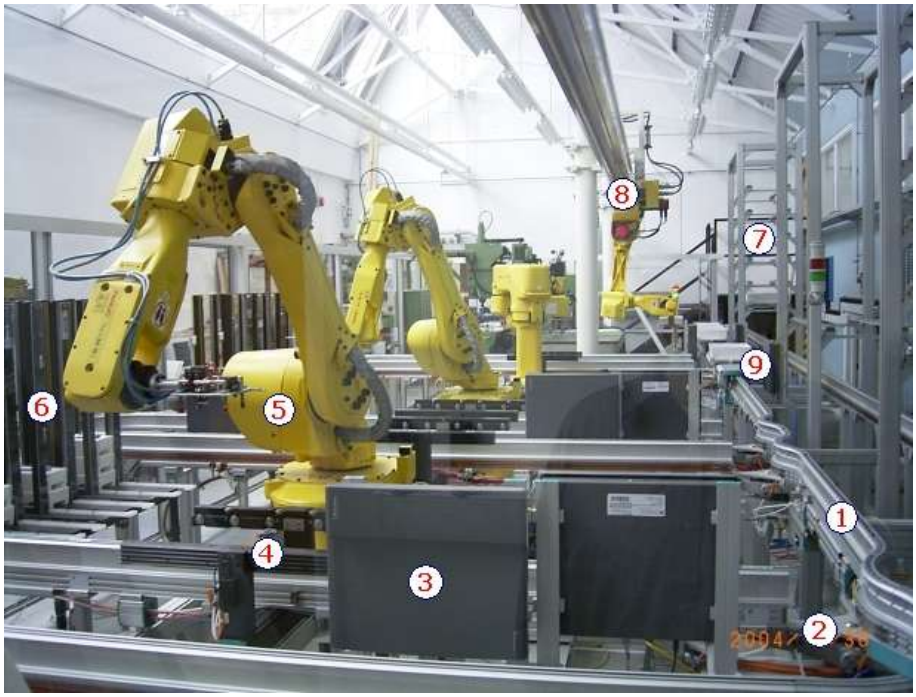

Fig. 3. The packing cell at the Center For Distributed Automation and Control University of Cambridge

3. RFID readers that read the EPCs of passing boxes – the data are provided by the readers to the gates to be able to properly navigate the shuttle.
4. Docking stations at which the shuttles are held while box is being packed.
5. Fanuc M6i robots that pack the boxes by the items picked up from the storage units.
6. Storage units for temporary holding of the items in four vertical slots (each for a particular type of the Gillette item).
7. Rack storages that hold shuttle trays with both the empty and packed boxes as well as with the raw items that can be used to feed the temporary storage areas.
8. Gantry robot agent that is able to pick the box out of the rack storage and drop it to the shuttle waiting in the docking station in the main loop.

It has been recognized, that the agent-based solution for the material handling tasks used in the MAST tool can be easily extended to provide the graphical simulation of the CDAC lab and, eventually, to be directly used for the physical control of the packing cell. The basic set of agents described in the previous Section has been extended with the following new agents:

a) **The RFID reader agent** that collects the EPC data from emulated RFID reader (one agent per one reader) and provides them to other agents via the FIPA's *subscribe interaction protocol* (see http://www.fipa.org/specs/fipa00035/ in (FIPA, 2004)). Generally, this mechanism allows any agent (let say A) to subscribe for being informed by other agent (B) each time a particular event happens at the B-agent. In the case of the RFID reader agent, there are two events about which the reader informs the subscribed agents: (i) the product with an RFID tag entered the range of the reader and (ii) the product leaved the reader's range. In both these cases the reader agent sends the FIPA's *inform* message including the EPC code(s) of the product to all agents that previously subscribed for one of these events.

b) **The gate agent** which is responsible for routing the shuttles at the crossing point of two conveyor loops. The navigation is based on the EPC code of the incoming box, which the gate agent obtains from the RFID readers located in front of the gate (previously described subscription mechanism is used). The issue is, that the EPC code, i.e. the 96-bit ID of the box does not directly contain the name of the destination docking station. To resolve this, we have introduced the product agent representing the shuttle (see bellow) that is registered in the agent platform under the same name as the ID stored in the EPC code of the box. As the product agent is aware of the destination docking station where it will be packed, the gate agent can query it for this information and then navigate it properly. Obviously, the gate agent is implemented as an extension of the diverter agent (see Section 3.2).

c) **The robot agent** that controls the packing operations performed by the Fanuc M6i robot. The packing operation starts by receiving a request from the product agent when the shuttle carrying the box reaches the docking station. The message sent to robot includes the specification of required items (e.g. two gels and one razor). The robot agent starts the negotiation with the associated **storage unit agent** to get the index of the slot from which the item of given type can be picked up (there are four vertical slots each for one type of the Gillette item). If the item is present in the slot, the robot picks it and places it into the box; if not present, the robot continues with the other required item. When all items are processed (either packed to box or missing in the storage unit), the robot agent informs the product agent that the operation has finished.

d) **The order agent** that is responsible for processing the user orders for packing the customized gift boxes. Automatically generated order agent (one per order) starts the negotiation with product agents that represent available shuttles in the system. One of the shuttles that is willing to cooperate (i.e. is not currently "working for" another order agent) is then committed the processing of the order.

e) **The product agent** representing a shuttle able to carry a box plays an active role in coordinating the packing operations. First it negotiates with the **rack storage agents** to find out if there is an empty box of user-requested type that could be used to fulfill the order. If so, the shuttle changes its destination to the docking station in the main loop where it stops and requests the **gantry robot agent** to pick up the box from particular rack storage and place it on the shuttle. Then there is another negotiation between the product agent and the storage units to select the one that holds all (or at least some) of the requested items. The shuttle then changes its destination again to reach the docking station that is next to the robot that will operate the selected storage unit. Any time the shuttle enters the RFID reader in front of the gate, the gate agent contacts the product agent representing the shuttle to get the name of its destination needed for navigation. Once the docking station is reached, the product agent requests the robot agent to pack the box (see the robot agent description above). When the processing of the box is finished by the robot, the product agent informs the order agent about the state of the order while releasing the shuttle from the docking station to return to the main loop. If the order is not fully completed, e.g. because there were not enough items in the storage unit, the product agent restarts the negotiation with the storage units again to obtain the remaining items. If the order is completed, the product agent negotiates again with the rack storages to select an

appropriate slot in racks where the packed box could be finally stored.

It is intended to use the MAST tool not only for the simulation but also for the physical control of the Cambridge packing cell. However, there are still some issues that need to be resolved in order to reuse the MAST agents in the implementation of the target control system. Particularly, as described in Section 3.1, the runtime interface between the agents and the emulation part of the MAST tool has to be changed so that the sensor and actuator data would be shared through the ControlLogix PLC tags. Another issue is the transfer of the EPC data from the physical RFID readers directly to the MAST's RFID reader agents – currently, we are developing the JAVA driver for accessing the RFID reader agent directly via the Ethernet.

4. CONCLUSION

In the case of multi-agent industrial solutions, under the term "simulation" we understand processes which are – in comparison to strictly hierarchical centralized systems – much more complex than just single simulation in the "classical" meaning. The simulation phase is much more crucial for the design and development process of agent-based systems than it has been for the development of "classical" centralized systems. It enables besides others:

- To **predict the behavior of the system as a whole**. The fact there is no central unit in the agent-based system represents a critical barrier in a wider applicability of the agent-oriented ideas. The simulation runs help to understand the system behavior and to detect the patterns of emergent/aggregate behavior. Considering that the behavior of a MAS is emergent, to ensure that all types of possible behavior were explored/covered by simulation still remains a painful problem (the situation is similar to that of system testing).

- To **predict and test the optimal scenario for the agent-based system** development.

- To select **the most optimal negotiation framework and strategy** for individual units in the system.

- To **directly link the simulation with real-life manufacturing/control processes**. That means that whereas a part of the agent-based system is engaged fully in the real-life activities, the remaining part can be just emulated. The shift of the borderline between the emulated and the real part of the system can be carried out in a quite smooth way. This would help to speed-up the initial "commissioning" process significantly.

One of the agent-based simulation tools for real-time manufacturing control, the Java-programmed MAST tool was briefly presented in this paper. This tool, similarly to other Rockwell Automation simulation tools for agent-based control, explores the universal run-time interface designed in such a way, that it is located inside the PLC controller and enables the real-time information transfer among several key components of the simulation environment, namely among the agent control, emulation of the physical equipment, physical equipment itself and the visualization module. This solution was found to be extremely efficient and fully suitable for industrial deployment.

REFERENCES

FIPA – Foundation for Intelligent Physical Agents organization (2004). website: http://wwww.fipa.org.

Fletcher, M., D. McFarlane, A. Lucas, J. Brusey and D. Jarvis (2003). The Cambridge Packing Cell – A Holonic Enterprise Demonstrator. In: *Multi-Agent Systems and Applications III* (V. Mařík, J. Müller, M. Pěchouček. (Ed)), pp. 533-543. LNAI 2691, Springer Verlag, Berlin, Heidelberg.

JADE agent platform (2004). website: http://jade.cselt.it.

Mařík, V., M. Pěchouček, P. Vrba and V. Hrdonka (2003). FIPA Standards and Holonic Manufacturing. In *Agent Based Manufacturing: Advances in the Holonic Approach* (S.M. Deen (Ed.)), pp. 89-121. Springer-Verlag, Berlin Heidelberg.

Maturana, F., R. Staron, K. Hall, P. Tichý, P. Šlechta and V. Mařík (2004). An Intelligent Agent Validation Architecture for Distributed Manufacturing Organizations. In: *Emerging Solutions for Future Manufacturing Systems* (L. M. Camarinha-Matos (Ed.)), pp. 81-90. Springer Science+Business Media, New York.

Parunak, H.V.D, R. VanderBok (1997). Managing Emergent Behavior in Distributed Control Systems. In *Proceedings of ISA-TECH/97 conference*. Anaheim, CA.

Vrba, P (2003). MAST: Manufacturing Agent Simulation Tool. *In proceedings of IEEE Conference on Emerging Technologies and Factory Automation*, **Vol. 1**, pp. 282-287. Lisbon, Portugal.