ISSN 1392 - 1207. MECHANIKA. 2011. 17(5): 551-556

# Agent-based methodology for developing mechatronic systems software

**L. Kizauskiene*, E. Kazanavicius**, R. Gaidys*****
**Kaunas University of Technology, Studentų st. 50, 51368 Kaunas, Lithuania, E-mail: losta@ifko.ktu.lt*
***Kaunas University of Technology, Studentų st. 50, 51368 Kaunas, Lithuania, E-mail: ekaza@ifko.ktu.lt*
****Kaunas University of Technology, Studentų st. 65, 51369 Kaunas, Lithuania, E-mail: rimvydas.gaidys@ktu.lt*

crossref http://dx.doi.org/10.5755/j01.mech.17.5.735

## 1. Introduction

Due to recent technological advances, the development of mechatronic systems shift from only integrating the mechanical, electrical, electronic and computer systems to an evolutionary development of systems, capable of complex decision making, adding intelligence to a mechanical design or even replacing mechanical designs with an intelligent electronic solution. Embedded systems play an important role in this process which is based on finding an optimal interconnection between the basic mechanical structure, sensors and actuators, information processing and control. The relation between the field of embedded systems and mechatronics becomes even more significant as the intelligence of embedded systems can bring value-added services for mechatronic devices as well as ensure other essential product characteristics such as reliability, enhanced performance, fail-safe operation and interoperability.

Growing requirements for modern embedded software alter their development methods, processes and the means for their construction. Reduced time to market, decreased development and operating costs, flexible design, fail-safe performance, predictable behaviour under hard real-time constraints, scalable and open architecture supporting software reuse and reconfiguration are only part of the challenges for researchers both in industry and academia. Furthermore, such systems are expected to be autonomous, collect environmental data, cause impact on other system elements, analyse complex situations and make decisions. On the other hand, it should be noted, that such system aspects as operating speed or optimal code become less important as modern computer systems ensure sufficient resources. Besides, evolving technologies such as cloud computing already contribute to shifting various services to the net, eliminating the need to implement them straight on devices. Consequently, such tendencies and evolving use of e-services only testifies one more important requirement for embedded design – assuring system interoperability.

Agent paradigm, acknowledged by the research community as a rising tool for solving complex problems [1, 2], can be a good alternative for developing intelligent embedded systems as it offers several advantages over traditional methods: high level of abstraction, easy system modification and extensibility, integrity, effective communication and other relevant features [3 - 6]. Thus it is reasonable to use the experience and the results of the ongoing research in this field and adapt an agent-based methodology for developing embedded applications for mechatronic systems domain. The selection of existing agent-based methodologies is broad [7, 8], however there exists no specialized methodology covering the required embedded systems domain criteria, nor the requirements for their development process. JADE – a java-based middleware [9] is chosen as the most suitable tool for the implementation of intelligent embedded systems that eases the development process assuring the features, required by every agent: message transport, behaviour scheduling, ontology, FIPA communication protocols, etc. Nevertheless, various domain specialists have to program specific applications from the beginning to the end. Therefore, this tool is insufficient to ease and accelerate the development of intelligent embedded systems as much as it could be done by reusing domain-specific components.

This paper presents an agent-based embedded control system design methodology for mechatronic systems. Embedded systems framework serving as a foundation for the methodology, has been created, tested and evaluated experimentally on prototype systems [10, 11]. It proved that the methodology simplifies and accelerates the implementation stage as well as the design stage of intelligent embedded system development process. Besides, it assures a high level of component interoperability and systems extensibility degree. Smart refrigerator control application, presented in this paper illustrates several steps of the proposed methodology and validates its feasibility for developing intelligent mechatronic system applications.

## 2. System design method

This section presents the basic steps of the proposed embedded system design process, which are further detailed by the proposed methodology, explaining the analysis, design, the integration, realization, testing and deployment stages.

According to the characteristics of intelligent embedded systems (IES) and the requirements to shorten their

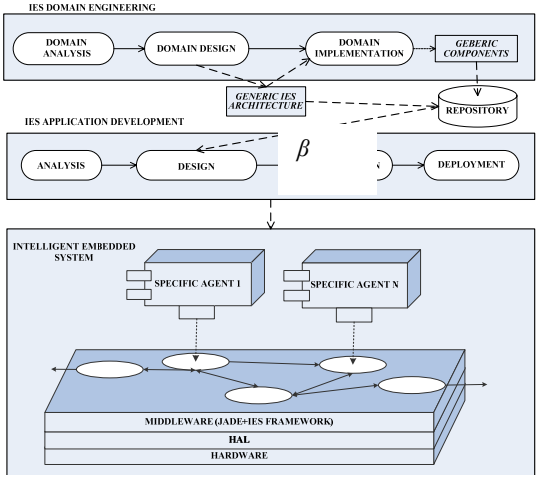


Fig. 1 Intelligent embedded system (IES) design process

development process, two separate processes are performed in their design process (Fig. 1):

- the development of system framework, including reusable generic IES architecture and generic agent-components, as IES domain engineering;
- the development of IES applications, based on the results, obtained during the first process.

The results of domain engineering process were presented in [10, 11], while the methodology for IES applications development is introduced below.

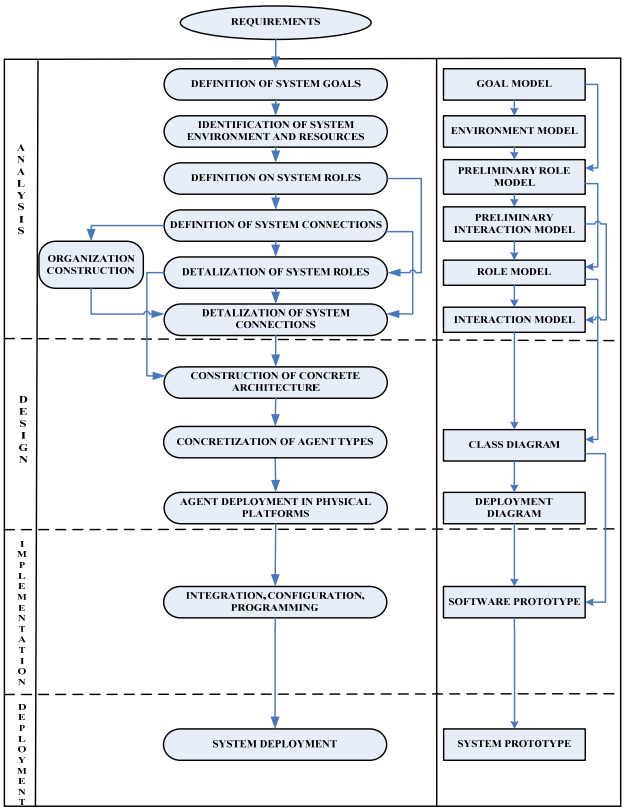


Fig. 2 Intelligent embedded system design methodology

The proposed IES methodology covers the analysis, design, implementation and deployment phases. The required steps and the results obtained in each of them are presented in Fig. 2. The goal of the analysis phase is to specify the system and its structure. It is achieved by specifying the system organization, which is perceived as a set of agent roles, their connections and communication patterns. Analysis phase results in the construction of certain models: goal, environment, roles and interaction models.

- Goal model. Based on system specification and requirements, the system is divided into separate subsystems and associated system goals are identified. Goals represent what the system has to achieve. However, talking about real time systems, the achievement of system goals depends on the requirements of time. IES are treated as soft real time systems, so, defining system goals requires specifying not only *what* the system is supposed to achieve, but also *when* the tasks should be performed.
- Environment model. In this step, the components of system environment are identified. The agent system is separated from the environment, where it resides, and the resources that it will be consuming are identified. Embedded systems by their nature are part of bigger systems, therefore the environment model is important because it represents environmental characteristics. Hardware sensors and actuators are treated as a part of environment and the data they provide are resources, read, consumed or modified by agents.
- Preliminary role model. Agent role defines the functionality and behaviour, which is expected from the agent type, performing the role. Preliminary system roles are identified according to the system goals. Preliminary role is specified by two attributes: responsibilities and permissions. Responsibilities are a key attribute that describes the functionality of a role. They are defined by taking into account three issues: system goals, communication between agents and interaction between agent and its environment. There are two types of responsibilities: liveness and safety. Safety responsibilities define the conditions that have to be maintained through the existence of a role, e.g. assure, that the temperature stays between 15 and 25 degrees. Liveness responsibilities are expressions, describing the sequence of activities and protocols, performed by a role. In order to fulfill the responsibilities, the role has a set of permissions, which state, what resources the role can read, modify or generate (Table 1).
- Preliminary interaction model defines the communication between system roles. It is defined by the parameters, depicted in Table 2.
- Agent organization. Organizational structure is one of the most important things in designing agent systems, because it outlines the coordination, subordination and control issues. Preliminary system role and interaction models identify system functionality independent of the system structure. Thus, the restrictions, applied to whole system are identified in this step.
- The preliminary role and interaction models are revised in this step for two reasons: firstly, the reusable component repository has to be taken into account; secondly – if the organizational structure, chosen in the previous step is different than identified in the proposed framework, it might affect the roles and communication models. The identified role and interaction models are compared to the specifications of agent library components and revised.

The analysis stage results in identifying the characteristics of intelligent embedded systems and their environment. These specifications are further used in designing generic IES architecture. This stage is based on a proposed design solution – embedded systems framework, which encompasses generic reusable components, their internal structure and generic architecture.

- The selection of the target application architecture is the first solution to be made in this stage. As the developers can use the proposed system framework, it is modified according to specifications, produced in the analysis phase. The identified agents are ‘substituted’ to the generic system architecture – thus obtaining the concrete application architecture. Besides, the developers have to take into account that the framework proposes certain system management agents, that have to be included in all specific architectures.
- Specific agent roles, identified in the analysis phase are associated with generic component library specifications, which lead to the identification of required system agent types and their internal structure. If there exists a possibility to choose, which roles can be associated with which agent types, it is left for the developer to make these

decisions.

- The result of this step is system deployment diagram. It is obtained from the system architecture and the developers only need to specify how system agents will be deployed on physical platforms.

In the stage of integration, realization and testing, the system is being implemented and tested. The implementation is performed by integrating the reusable agent components. Some components can be reused without modifications, others – with setting required parameters or little additional programming for adding specific service logic. Agents are integrated, configured, executed and tested on the same programming environment – JADE platform, complemented by domain specific components. In the stage of the deployment, the designed system together with the environment, where it is implemented and tested is deployed on a selected physical platform.

## 3. Developing an intelligent inventory management agent for the smart fridge system

This section presents a smart refrigerator control application, created using the proposed methodology, more focusing on intelligent inventory management agent as several implementation details of intelligent fridge application have already been presented in [11]. Creating the smart fridge control application, several goals were defined at the analysis stage: maintaining the proper temperature, controlling the light, monitoring the products as well as optimization and planning of product orders. The IES framework enabled the reuse of the whole temperature control application, implemented during another experiment [10]. Thus, only the last two goals were analyzed further and lead to identification of required agent types – namely sensor agent for RFID sensor; information processing agent for implementing the role of a product monitor agent, as well as intelligent decision agent – for realizing the role of product planning and ordering agent.

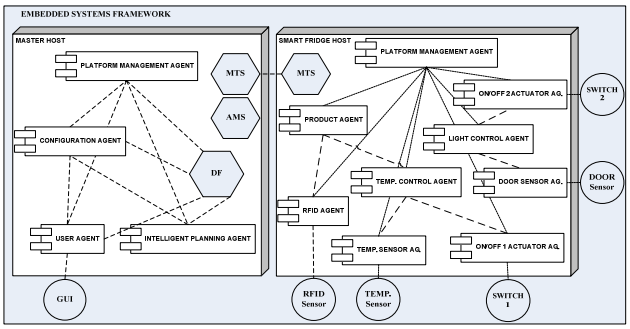


Fig. 3 Smart fridge control application deployment architecture

The smart refrigerator system architecture and its deployment scenario are depicted in Fig. 3. The system is deployed on two hosts. One of the hosts is a master host which is responsible for the platform and configuration management, another one is smart fridge control host. Master host also runs agents responsible for interaction with the end-user. Intelligent inventory planning agent is deployed on master host as well.

The smart fridge control host is a small embedded device, running embedded Java Virtual Machine. Every sensor/actuator is associated with corresponding agents. Main services, associated with appropriate service agents, include: product management, fridge temperature control, light control and user notification. Product management service includes automatic product identification through RFID sensor, as well as product amount monitoring and absence detection. Fridge temperature control agent is a service agent which has established connections to temperature sensor and on/off switch 1. This agent periodically measures temperature and adjusts heating switch state according to the control algorithm. Light control service is responsible for triggering the light switch (on/off switch 2), when the associated door sensor indicates that the fridge door is open/closed. The system is designed to warn the user when the product expires, the amount of the product is getting low, or in case, the door of the fridge is left open for a longer time.

Inventory management is an important issue for most wholesale and retail companies, as they need to maintain the inventories of goods to be available for purchase as well as reduce their storage costs. Planning and scheduling, ensuring that the needed materials arrive "just-in-time" for their use, should be an important part of companies' inventory policy. Smart fridge product monitoring and automatic ordering tasks correspond to the problems, handled by the inventory theory applications.

The goal of inventory management agent is to predict an optimal amount of the required products, the time and the size of the order, taking into account the reliance between expenses on product price, their amount, storage and delivery. The functionality, behavior as well as communication of the inventory planner agent are depicted in the role (Table 1) and interaction models (Table 2).

Table 1

| **Role** | Inventory planning agent |
|---|---|
| **Description** | Collects product consumption statictics, analyses data and decides what orders should be completed. |
| **Protocols and activities** | PublishEvent, PublishAction, PublishServiceInDF, RegisterToEvents, GetConsumptionData, ProcessInformation, SaveData, UpdateData, GetOrderQuery, AnalyzeData, MakeDecision, SendOrderProposal. |
| **Permissions** | *Read* Message content, Product data, Agent state<br>*Create* Consumption statistics<br>*Change* Agent state, Consumption statistics |
| **Responsibilities**<br>**Liveness** | INVENTORY PLANNER = (ConfiguresItself).(CollectsConsumptionStatistics)\|\|(Computes).<br>ConfiguresItself = PublishesEvent. PublishesAction. PublishesServiceInDF. ConnectsDeviceAgents. RegistersToEvents.<br>CollectsConsumptionStatistics = GetConsumptionData. ProcessInformation. SaveData. UpdateData.<br>Computes = GetOrderQuery. AnalyzeData. MakeDecision. SendOrderProposal. |
| **Safety** | Always ensure communication with data providing agents. |

Table 2

| **Protocol** | *SendQuantity Request* | *SendProduct Data* | *SendConsumptionData* |
|---|---|---|---|
| **Initiator** | Product manager agent | RFID Sensor agent | Product manager agent |
| **Responder** | Inventory planner agent | Product manager agent | Inventory planner agent |
| **Responding action** | SendOrder Proposal | | |
| **Goal** | After getting the quantity request, inventory planner agent analyses the data and computes the required amount of products. | RFID sensor agent reads product data putting it in and taking it out of the fridge and sends the data to the product manager agent. | Inventory planner agent registers to the event for getting the information about consumed products and after getting it, saves the data to the product consumption registry. |

Inventory planner agent service logic and decision making mechanism. Modern companies use scientific inventory management systems that are usually based on operations research and comprise the following steps:

1. formulation of a mathematical model describing the behaviour of the inventory system;

2. **s**eeking an optimal inventory policy with respect to this model;

3. using a computerized information processing system to maintain a record of the current inventory levels;

4. using this record of current inventory levels, applying the optimal inventory policy to signal when and how much to replenish inventory.

Service logic and the decision making mechanism of inventory planner agent chooses an appropriate mathematical model, depending on the given situation and the input data, making an optimal decision, when and how to replenish the products in the fridge. The agent logic is implemented based on three mathematical inventory management models: economic order quantity model (EOQ), EOQ model with planned shortages or EOQ model with quantity discounts [12]. The first model is presented below for more details.

The most common inventory situation faced by manufacturers, retailers, and wholesalers is that stock levels are depleted over time and then are replenished by the arrival of a batch of new units. A simple model representing this situation is the following economic order quantity model or, for short, the EOQ model.

Units of the product under consideration are assumed to be withdrawn from inventory continuously at a known constant rate, denoted by a; that is, the demand is a units per unit time. It is further assumed that inventory is replenished when needed by ordering (through either purchasing or producing) a batch of fixed size ($Q$ units), where all $Q$ units arrive simultaneously at the desired time. For the basic EOQ model to be presented first, the only costs to be considered are:

- $K$ - setup cost for ordering one batch,
- $c$ - unit cost for purchasing each unit,
- $h$ - holding cost per unit per unit of time held in inventory.

The objective is to determine when and by how much to replenish inventory so as to minimize the sum of these costs per unit time.

We assume continuous review, so that inventory can be replenished whenever the inventory level drops sufficiently low. We shall first assume that shortages are not allowed. With the fixed demand rate, shortages can be avoided by replenishing inventory each time the inventory level drops to zero, and this also will minimize the holding cost.

To summarize, in addition to the costs specified above, the basic EOQ model makes the following assumptions.

1. A known constant demand rate of units per unit time.

2. The order quantity (Q) to replenish inventory arrives all at once just when desired, namely, when the inventory level drops to 0.

3. Planned shortages are not allowed.

In regard to assumption 2, there usually is a lag between when an order is placed and when it arrives in inventory. The amount of time between the placement of an order and its receipt is referred to as the lead time. The inventory level at which the order is placed is called the reorder point. To satisfy assumption 2, this reorder point needs to be set at the product of the demand rate and the lead time. Thus, assumption 2 is implicitly assuming a constant lead time. The time between consecutive replenishments of inventory is referred to as a cycle. The total cost per unit time $T$ is obtained from the following components. Production or ordering cost per cycle is equal to

$$K + cQ \tag{1}$$

The average inventory level during a cycle is $Q/2$ units, and the corresponding cost is $hQ/2$ per unit time. Because the cycle length is $Q/a$, Holding cost per cycle is equal to

$$\frac{hQ^2}{2a} \tag{2}$$

Therefore, total cost per cycle is equal to

$$K + cQ + \frac{hQ^2}{2a} \tag{3}$$

so the total cost per unit time is

$$T = \frac{K + cQ + hQ^2/(2a)}{Q/a} = \frac{aK}{Q} + ac + \frac{hQ}{2} \tag{4}$$

The value of $Q$, say $Q^*$, that minimizes $T$ is found by setting the first derivative to zero

$$\frac{dT}{dQ} = -\frac{aK}{Q^2} + \frac{h}{2} = 0 \tag{5}$$

so that

$$Q^* = \sqrt{\frac{2aK}{h}} \tag{6}$$

which is the well-known EOQ formula. The corresponding cycle time, say $t^*$, is

$$t^* = \frac{Q^*}{a} = \sqrt{\frac{2K}{ah}}\ . \tag{7}$$

## 4. Inventory planning experiments and results

During the first experiment, the agent is given such initial data: it is known that the demand for product A is about 100 units every day. The setup cost for ordering one batch is 100 LTL. The holding cost per unit is 0.02 LTL per day. Having this data the agent has to inform about the ordering time of product A and the size of the batch, considering the time between the placement of an order and its receipt (lead time) is 12 days. From the given data, the agent identifies that for calculating the order cycle and the size of the batch it has to use the EOQ model.

The optimal order size $Q^* = 1000$ units, while the length of the cycle $t^* = 10$ days. As the lead time is 12 days and the cycle of the order is 10 days, the new order has to be placed when the remainder of product A is enough to satisfy its demand for 2 days. Thus, 1000 units of product A are ordered, when its remainder is 200 units (Fig. 4). After getting the message about the remainder of product A, the agent analysis the message and decides upon the necessity to make a new batch order and informs the user about the taken decision.

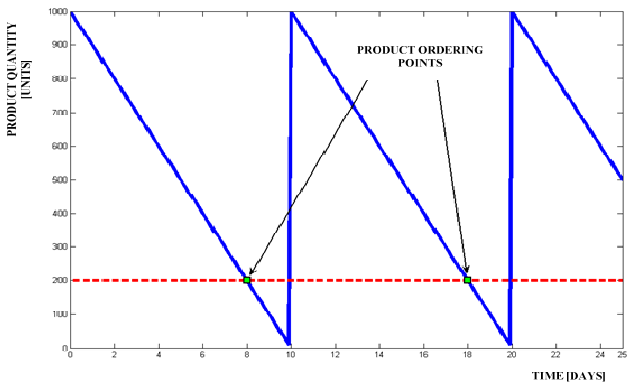


Fig. 4 Product order prediction using EOQ model

In the next experiment, the agent is given such initial data: the demand for product C is 100 units every day. The setup cost for ordering one batch is 100 LTL. The holding cost per unit is 0.02 LTL per day. The price for one unit, depending on the size of batch is: 11 LTL, if the order is 500 units; 10 LTL, if the order size is between 500 – 7000 units and 9.5 LTL, if the order size is more than 7000 units. Having this data and evaluating the cost for holding the inventory, the agent has to inform about the optimal batch size for product C. From the given data, the agent identifies that it has to apply the EOQ model with quantity discounts.

The optimal order size here is *Qopt* = 1000 units, which is enough for 10 days, while summing up the hold-

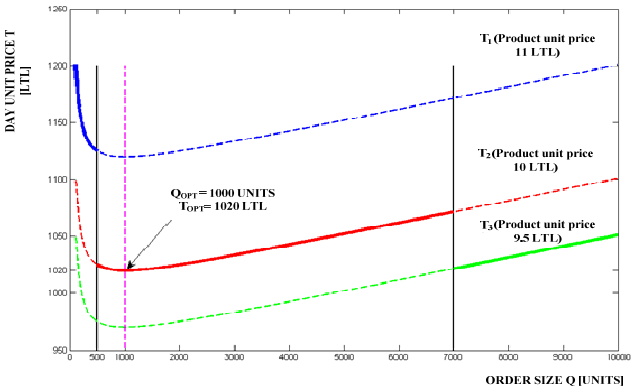


Fig. 5 Product order prediction using EOQ model with quantity discounts

ing price for one day, *Topt* = 1020 LTL (Fig. 5). If the price of the good would decrease up to 9 LTL, when ordering more than 7000 units, then the batch size would be $Q^*3 = 7000$ units, which would be enough for 70 days and the price for one day would be $T3 = 971.4$ LTL. Similar situation would be if the size of the batch, where a unit price is 9.5 LTL could be decreased by 100 units (Fig. 5). Consequently, after getting the message about the necessity for good C reorder, the inventory planner agent analyzes the data and makes the decision about the optimal amount of the order.

## 5. Conclusions

The proposed methodology simplifies and accelerates the implementation as well as the design stage of embedded system development process. It assures a high level of component interoperability and systems extensibility degree. Contrary to existing agent-based methodologies, it addresses specific embedded systems characteristics – specification of system boundaries, the modelling of goals and time requirements; specific design process requirements – the adopted system development life-cycle enables the redesign and gradual concretization of generic design artefacts, assuring the reuse of generic agent components and generic IES architecture.

## References

1. **Ning, K.J.; Yang, R.Q.** 2006. MAS based embedded control system design method and a robot development paradigm, Journal of Mechatronics, vol. 16: 309-321.
2. **Burmester, S.; Giese, H.; M¨unch, E.; Oberschelp, O.; Klein, F.; Scheideler, P**. 2008. Tool support for the design of self-optimizing mechatronic multi-agent systems, International Journal on Software Tools for Technology Transfer (STTT), vol. 10, number 3: 207-222, DOI: 10.1007/s10009-008-0067-0.
3. **Xinglu, M.; Yingjie, Q**. 2008. Research on embedded agent system architecture, Proceedings of the 2008 International Conference on Embedded Software and Systems Symposia. Washington, DC, USA.
4. **Jamont, J. P.; Occello, M.** 2007. About some specificities of embedded multiagent system design. The diamond method. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Silicon Valley: US.
5. **Povilionis, A.; Bargelis, A.** 2010. Structural optimiza-

tion in product design process, Mechanika 1(81): 66-70.
6. **Čikotienė, D.; Bargelis, A.** 2009. Research of quality impact to the product design properties and characteristics, Mechanika 5(79): 63-67.
7. **Zohreh Akbari, O.** 2010. A survey of agent-oriented software engineering paradigm: Towards its industrial acceptance, Journal of Computer Engineering Research, vol. 1(2): 14-28.
8. **Bergenti, F.; Cleizes, M.-P.; Zambonelli, F.** 2004. Methodologies and Software Engineering for Agent Systems. The Agent-Oriented Software Engineering Handbook, Kluwer.
9. **Bellifemine, F.; Caire, G.; Greenwood, D.** 2007. Developing multi-agent systems with JADE. Wiley Series in Agent Technology. ISBN 978-0-470-05747-6. February.
10. **Kazanavicius, E.; Kazanavicius, V.; Ostaseviciute L.** 2009. Agent-based framework for embedded systems development in smart environments. In Proceedings of International Conference on Information Technologies (IT 2009), Kaunas.
11. **Kazanavicius, E.; Kazanavicius, V.; Ostaseviciute, L**. 2009. A reusable agent-based framework for smart embedded systems. Solid State Phenomena: Mechatronics Systems and Materials III: selected, peer reviewed papers from the 5th international conference MSM 2009, Vilnius, Lietuva.
12. **Chan, Alan H. S.; Ao, Sio-Iong**. 2008. Advances in Industrial Engineering and Operations Research. Springer. ISBN 978-0-387-74903-7.

L. Kizauskiene, E. Kazanavicius, R. Gaidys

AGENTINĖMIS TECHNOLOGIJOMIS PAGRĮSTA MECHATRONINIŲ SISTEMŲ PROGRAMINĖS ĮRANGOS KŪRIMO METODIKA

R e z i u m ė

Straipsnyje pristatoma agentinėmis technologijomis pagrįsta mechatroninių sistemų programinės įrangos kūrimo metodika. Pateikiamas eksperimentinio šios metodikos taikymo pavyzdys – intelektinės išmaniojo šaldytuvo sistemos prototipas.

L. Kizauskiene, E. Kazanavicius, R. Gaidys

AGENT-BASED METHODOLOGY FOR DEVELOPING MECHATRONIC SYSTEMS SOFTWARE

S u m m a r y

This paper presents an agent-based embedded control system design methodology for mechatronic systems. The methodology is illustrated by an intelligent fridge prototype system.

Received March 03, 2011
Accepted October 21, 2011