BESA-ME: Framework for Robotic MultiAgent System Design

David M. Flórez, Guillermo A. Rodríguez, Juan M. Ortiz and Enrique González

SIDRe Research Group, Pontificia Universidad Javeriana, Bogotá, Colombia

Abstract. BESA-ME is a software middleware designed to make easier and to improve the construction of robotic control systems based on multi-agent techniques. BESA is a behavior-oriented, event-driven and social-based general purpose architecture designed to build concurrent applications using the multi-agent paradigm. The BESA abstract model incorporates the concept of behavior and the management of asynchronous events, which are very useful in the construction of robotic systems, thus it allows to design robot control architectures in a natural and direct fashion. BESA-ME, micro-edition, is the adapted BESA model that is well suited to be implemented over microcontrollers in embedded systems. Initially, it has been developed for the PIC18F chip family, and then adapted for dsPIC60F chip family, both under the real time operating system FreeRTOS ™.

1 Introduction

When dealing with the construction of complex systems, a design methodology usually includes splitting the system in a set of smaller simple tasks. This approach allows dealing with complexity and makes possible to improve the efficiency of the system implementation by distributing the execution of these tasks on a set of processing units. The agent paradigm provides a conceptual framework where systems can be analyzed and synthesized as a collection of interacting entities, using a high level abstraction, which fits in a natural fashion with the implicit concurrent requirements of robotic systems.

The development of new products implies the incorporation of innovative and more complex functionalities. The actual tendency of the technology used to improve the system efficiency is to construct multi-processor, multi-core or multi-thread machines [1]. This approach has the advantage of being easily scalable; however, in this case it is necessary to include specialized mechanisms to solve synchronization and communication problems. In addition to the high computational power requirements, some applications like smart home systems [2], robotics, automation, industrial machines and multi-robot systems [3] require that the embedded hardware communicates frequently with external stand alone systems. As a consequence, design of software for embedded systems must: generate modular concurrent solutions to deal with complexity, take advantage of hardware parallelism, and be adequate to interact in a natural fashion with external stand alone systems. It's

requirements to have models and platforms that ease the system design, taking advantage of the parallel processing capabilities of the actual and future processors.
In the context of robotic systems, the design problems previously introduced are amplified due to the fact that robots must be able to evolve in a not completely observable environment under real time constrains. In order to deal with these critical conditions, the designer must also take into account the following issues:

- manage complex synchronization of non-deterministic events coming from sensors and controlled actuators.

- distribute the application into several processes, using communication taking advantage of the parallel and specialized hardware.

- use an unified model in both the embedded and the external components of the system.

- use of a holistic approach where the robot control unit is modeled as a unique composed system; the system is seen as a collection of logical units, which can be physically deployed in the available hardware (embedded or not).

The actual software tools to model, design and implement embedded distributed and real-time systems covers different needs. Real Time Operating Systems (RTOS) are used for the implementation of systems with time-response constraints [9]. For instance, TURTLE [6] offers an environment based on the Unified Modeling Language (UML) for the formal model [7]. Another interesting approach uses the synchronous language LUSTRE, designed for the development of critical control software [8]. These approaches are process oriented, where the basic processing units are processes or tasks that usually communicate by message passing mechanisms. Even if the conceptual model provided by the notion of process is very useful, general and flexible, it has a low level of abstraction, which makes it difficult to be used in the design of complex systems. For instance, when designing a robot or multi-robot system, it would be preferable to use a conceptual model with higher degree of abstraction, where notions as behaviors and goal-oriented entities could be modeled in a more direct way.

A well suited approach to model complex problems is the Multi-Agent System (MAS) paradigm. Different activities can be distributed into several cooperative autonomous entities, and interactions are the basis for the dynamics of the system. Agents respond to events coming from its environment or derived from its interactions with other agents. The communication is the basis to construct the social level which emerges from the inter-agent interaction [10]. A MultiAgent System, MAS, is a computational cooperative system capable of executing concurrent tasks through its agents.

BESA is a MAS architecture [5] [17] that aims to solve the problems that where depicted in the precedent paragraphs. BESA provides an abstract model to construct multi-agent systems. The initial implementation of the BESA model was developed to work in a Java distributed environment. The BESA micro-edition, BESA-ME, is introduced in this paper. The BESA-ME model, architecture and implementation have been developed for embedded systems using RTOS as software support, running in microcontrollers and DSP hardware platforms. This development is mainly motivated to deal with the requirements involved in the design of multirobot systems, where a recursive organizational approach is applied.

In this paper the BESA-ME conceptual model and architecture are explained. Then, the RTOS software and hardware considerations are analyzed in order to adjust the model to fit the constraints of a practical embedded application. Finally, the implementation strategy is depicted and the obtained results are analyzed.

2 BESA Architecture

The BESA architecture defines a conceptual model for the implementation of an agent framework. The construction of a complex and concurrent application must use this agent conceptual abstraction to model the system. Then the implementation of the system can be performed in a direct fashion exploiting the facilities provided by the BESA application framework.

2.1 Abstract Model

The BESA abstract model is supported by three principal concepts, which provide a theoretical frame that integrates a behavior-oriented, event-driven and social based approach in a coherent structure.

- Behavior-oriented: agents are composed by a collection of behaviors, simple entities charged of the treatment of a set of related events.
- Event-driven: asynchronous events unleash the behavior execution. They represent signals that could be perceived by the agents. The behavior execution is controlled determined by a guard based selector mechanism; guards forbid events to be processed if a desired condition is not attained.
- Social-based: the multi-agent system is created to form a social organization, where well-known communication patterns can be used; it is also possible to utilize mediator agents that help in the correct development of interaction protocols between agents.

BESA is a concurrent oriented architecture; agents are internally seen as a multithread system and the non-determinism, implicit in an event driven system, is managed by a select (alting) mechanism.

2.2 Agent Level

The agents give a response to events coming from the environment or from other agents. BESA events have a specific semantics and are marked with an event type label. The agent has an associated treatment for each type of event. The treatment of an event must include the rational response of the agent. When processing an event, the data attached to the event and the internal state must be taken into account. The response to an event can be produced by any kind of decision mechanism (neural networks, fuzzy logic, procedural code, rule based system, etc.). This response can include: the selection of the appropriated actions that must be executed, sending a new event to other agents, and the modification of the internal state of the agent. Events exchanged between agents usually follow well-defined communication patterns, known as interaction protocols.

A BESA agent is structured with three main components: the channel, a set of behaviors and the agent state. Figure 1 shows these components and the way they are connected. The channel manages a mailbox where events are received, providing the unique entry point to the agent. The received events are assigned to different ports depending on their event type. Finally, they are transferred to the correct behavior, if the guard condition is verified.

The behavior is an execution space that is activated when an associated event is received. Then the corresponding treatment is executed, thus producing the rational response to the received event. An agent can have several concurrent behaviors, thus allowing to process several events at the same time. Notice that the way that the parallel execution is performed depends on the capabilities of the executing environment (hardware and software). Incoming events are received by the channel and transferred to a queue in the appropriated behavior; events received by the same behavior are processed in a sequential order. A guard verifying function and a treatment function are associated to each type of event.

The agent state provides a mechanism to store information about the agent, the environment, other agents and the global system; it is used to keep data in a persistent way. This information usually is used in the treatment of events. The state is a structured shared memory; thus, the concurrent access to this space must be synchronized. The synchronization could be implemented with semaphores or other operating system mechanisms.

2.3 Social Level

The social level aims to provide a set of predefined mechanisms and interaction protocols that could be directly used to manage agent interaction. Some of the BESA cooperation and communication services provide specialized ready to use agents that act as mediators in the social organization. For instance, the group communication service uses a mediator agent, actuating as a router, distributing events in the same order to all the agents subscribed to the group. A more detailed presentation of the social level is out of the scope of this paper. In [17], there is a more detailed description of this level.

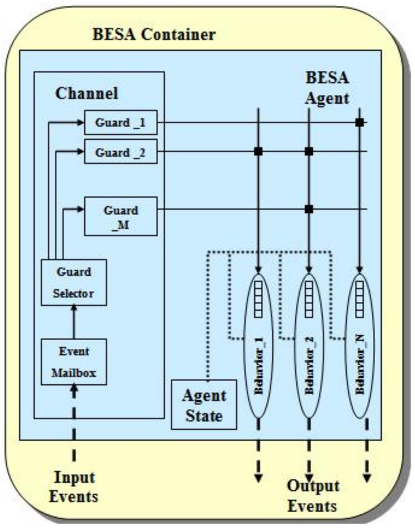

Fig. 1. Internal structure of a BESA Agent.

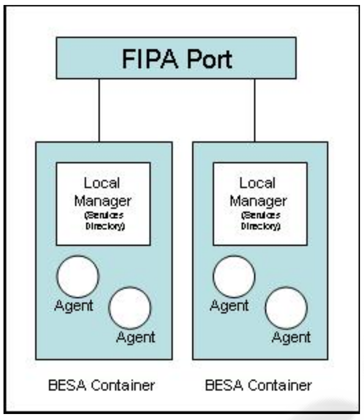
FIPA Port
Local Manager (Services Directory)
Agent
Agent
BESA Container
Local Manager (Services Directory)
Agent
Agent
BESA Container

Fig. 2. BESA System Layer Model.

2.4 System Level

The system level is formed by a set of agent containers running in physical or virtual machines. The container can be seen as the execution space for the agents (Figure 2). A BESA container has a local manager in charge of handling the agent's life cycle and the directory services. The container also assures the correct communication between agents "living" in different containers. The system level is designed to comply with the FIPA standard [11]. The inter-agent interaction in the same container is performed without the local manager intervention.

In order to make easy the communication between agents, BESA includes a directory service. The white pages component allows to locate agents by an ID. The yellow pages component makes possible to locate a group of agents that can provide a specific service.

3 BESA-ME DESIGN

The goal of the BESA-ME design is to find how to implement the BESA model in an embedded system, taking into account the BESA requirements and the constraints of this kind of platforms. The more important operating requirements of a BESA framework are: the management of the shared resources and shared memory spaces, and the multi-task operation and the concurrent treatment of events.

Real time operating systems are the proper tool to provide the services required to implement BESA-ME. In particular, the communication and synchronization of the agent behaviors requires semaphores and messages queues with blocking sending. The concurrent operation of agents and their behaviors can be achieved by a multitask preemptive operating system kernel. As BESA-ME is a framework for the construction of embedded applications, usually implemented in micro- controllers and DSPs, it is convenient to use a Real Time Operating System (RTOS) to provide the required functionalities [5]. A BESA-ME application can be seen as a layered structure, as shown in Figure 3. Upper layers use the abstraction and services provided by the lower layers. The user application is designed using the abstraction of the BESA abstract model. The BESA-ME components are implemented as concurrent tasks using the inter-task communication mechanisms provided by the RTOS.

Finally, it is essential to adjust the BESA general model in order to fit the constraints of a practical embedded application. In the BESA-ME implementation model, some BESA elements are excluded in order to improve the performance and reduce the amount of required memory. The conditions of the guard selector mechanism are eliminated, thus reducing the control of the non-determinism implicit in an event driven system, but improving the filtering speed of events in the channel and allocating more memory to store events in the ports. In the container only the white pages directory is used, so the agents can locate an agent if they know its "alias".

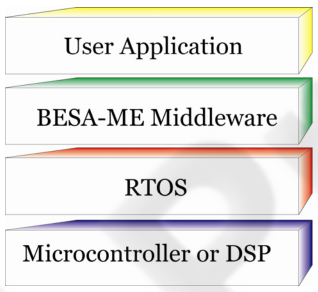

Fig. 3. Layers in a BESA-ME application.

4 Implementation

BESA-ME has been successfully implemented for the microcontroller family PIC18F and the DSP dsPIC60F. The available RTOS that were selected as candidates to support the BESA-ME implementation include the Salvo-RTOS [12], the µC/OS-II [13], the CMX-TINY [14] and FreeRTOS [15]. Though several of the studied RTOS were offered the required services, after a detailed analysis of their functionalities, the FreeRTOS™ was selected. It was chosen because it's a RTOS with a GNU license, it is well-structured and easy to use; it also supports many microprocessors, micro-controllers and DSPs.

4.1 Agent Level

The BESA agent is modeled by a data structure called stAgentBESA. It contains pointers to the agent shared state and channel, an array of pointers to the behaviors associated to the agent and the agent handler that contains the alias and the agent id.

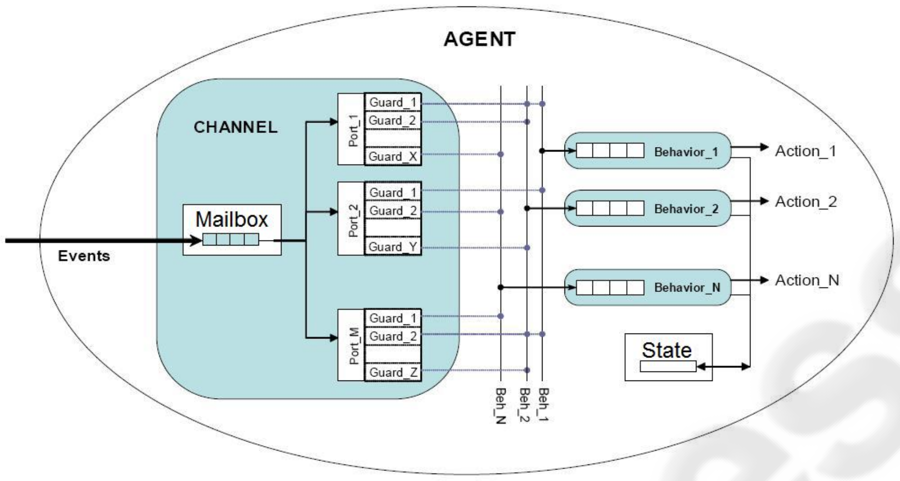

Fig. 4. BESA-ME agent level model. The boxes represent the data structures of the agent.

The Channel Task

The Channel Task has been implemented as a RTOS task, which is blocked waiting for incoming events. This waiting mechanism is implemented using a RTOS message queue service. When an event is received in the *Channel Reception Queue* this task looks for the incoming event type and compares it with the defined registered ports; if a match was found, the channel task sends a message to the behaviors interested in this event type. The same code is used to implement and create the required channel instances thanks to the parametric data structure used to represent an agent. When an agent is built, a channel task is created. A similar approach is used for the behaviors tasks. The channel parameters are contained in a data structure that includes a pointer to the channel reception queue, an array of pointers to the channel ports and some control variables.

The Behavior Task

The behaviors have been implemented as RTOS tasks, which are blocked waiting for messages sent from the channel. The messages contain the event attached data and a pointer to the adequate treatment function. The association between event types and their treatment functions is defined when the guards are created and bind to ports.

When a message is received, the adequate treatment function is executed; it receives the event data and a pointer to the agent state. The state is a shared memory that has a read/write protection mechanism built with a binary semaphore. In the treatment function the microcontroller peripherals are used, mathematic operations are performed, the external hardware is controlled, events to other agents are sent, and the logic that implements the rationality of the agent is included.

4.2 System Level

An agent system, implemented using BESA-ME is a distributed system composed by one or more BESA containers running in one ore more physical or virtual machines (Figure 5). In BESA-ME, there is only one container running in each processing device (microcontroller or DSP). Each container has a unique ID registered in the white pages directory. When a send event service is invoked, a search for the address of the agent is performed. If the destination agent exists in the local container, the event is directly sent to the appropriated message queue. If the destination is not local, the event is sent to the other registered containers; only the container where the receiver agents exist takes into account the event and transfers it to the appropriated agent.

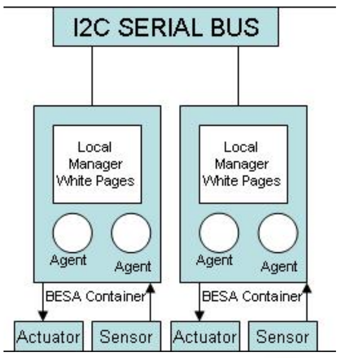

Fig. 5. BESA-ME System Layer Model.

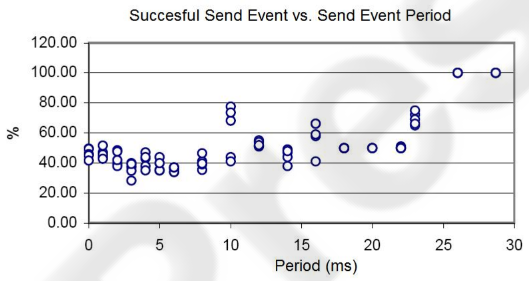

Fig. 6. Percentage of events received successfully by an external agent.

In the actual implementation of BESA-ME, the communication between containers is achieved through the use of a wired bus and the I2C low level communication protocol. The I2C protocol has been implemented in an interrupt service routine. The BESA-ME communication process is controlled with the I2C control bits at the physical layer and a BESA acknowledge to inform if the event has been taken properly by the destination agent. The information required to construct event and acknowledgment packets is written in a transmit buffer protected by a binary semaphore. The semaphore is free when the whole message has been send. Another semaphore prevents that one other local agent start an external communication before the reception of a pending acknowledgment packet.

5 Experimental Results

In order to verify that the BESA-ME execution is correct, a test protocol has been designed. The controlled independent variables include the task's stack size, the number of BESA-ME elements (agents, behaviors, guards) and the event production frequency. The measured dependent variables include the response time of the system to extern and local events, the number of errors in the inter-agents communication, and the evaluation of the general performance of the system.

The functional test was implemented in a couple of interconnected microcontrollers PIC18F8720 and PIC18F8620, with a processor frequency of 4 MHz and the I2C communication frequency of 100 KHz. This value of the intervenient variables were found by gradual increments. In the agent level the maximum number of behaviors is 11, with only one agent in the container. For one behavior the maximum number of guards is 20, with only one port (one event type). The maximum number of ports is 50. In the system level the maximum number of agents per container is 5. An extensive communication test was applied to the BESA-ME system level [16]. In summary, the results include the response time to events and the errors rate, in local and extern communication. In figure 6 can be observed how the event frequency affects the communication between agents placed in different containers.

The AgentCoop project aims to build a multirobot platform. This multirobot system is controlled using the MRCC model (MultiResolution Cooperation Control), which provides a flexible framework to built cooperative multiagent systems. In order to deal with the complexity of this application, the AgentCoop architecture has been designed using the BESA conceptual model. The robotic system is composed by a set of agents that can be deployed in a distributed environment, where high performance stand alone computers coexist with embedded processors The design of the AgentCoop platform represents a complex problem with the characteristics previously analyzed in section 1. The robots of the AgentCoop project are controlled using the dsPIC30F6012. Thus, BESA-ME has been easily migrated to work with this platform, proving the flexibility and modularity of the actual BESA-ME framework.

6 Conclusions

BESA-ME is a useful tool to solve complex problems in embedded applications. It allows to split complex objectives into simpler tasks, and to allocate them to dedicated agents. BESA-ME solve the low level synchronization problems, providing a high level abstraction model to make a more easy design and implementation of system. The use of the BESA-ME model allow the development of MAS oriented distributed applications for microcontrollers in a easy and efficient way, reducing the development time.

One of the more important BESA-ME facilities is the flexibility; if a system could not be fitted in only one microcontroller it can be distributed in two ore more processors only by changing configuration and declaration parameters of the middleware. The required external communication between micro-controllers is automatically detected and managed by the BESA-ME communication services.

A comparative evaluation of BESA-ME is projected. This evaluation will be implemented comparing the AgentCoop development model for BESA-ME and the implementation model if using only an RTOS without BESA-ME.

References

1. INTEL Corporation. “Intel® Dual-Core Processors: The first step in the multi-core revolution”, www.intel.com/technology/computing/dual-core. January 15th 2007.
2. SMARTHOME. “Getting Started”, http://www.smarthome.com/starters.html. January 13th 2007.
3. The Robotics Institute. “Distributed Robots Architecture”, Carnegie Mellon University. http://www.frc.ri.cmu.edu/projects/dira. January 12th 2007.
4. Liu, Jiming. “Multi – Agent Robotic Systems”. CRC Press, Boca Ratón, Fl. 2001.
5. Gonzalez Enrique, Avila Jamir, Bustacara César. "BESA: Behavior-Oriented, Event-Driven And Social-Based Agent Framework". PDPTA'03, Las Vegas-Usa, Csrea Press, Vol. 3, Junio 2003, PP 1033-1039. ISBN 1-892512-43-2.
6. P.De Saqui SanneS. “Conception basée modèle des systèmes temps réel et distributes”, Rapport Laas No05403, Toulouse, France. 7 Juillet 2005.
7. Roques, Pascal, “UML en action”. Editions Eyrolles, France. June 2004.
8. Verimag. “Synchrone”, http://www-verimag.imag.fr/SYNCHRONE/. October 2006.
9. Stallings, William. “Operating Systems: Internals and Design Principles“. Prentice Hall. USA. 2001.
10. Weiss, Gerhard. “Multiagent systems: A modern approach to Distributed Artificial Intelligence”. MIT Press, U.S.A 1999.
11. Foundation for Intelligent Physical Agents. “FIPA Specifications”, http://www.fipa.org, November 5th 2004.
12. PUMPKIN INC. “Salvo User Manual”, http://www.pumpkininc.com/, October 23rd 2004.
13. MICRIUM. “µC/OS-II Kernel Benefits”, http://www.ucos-ii.com/products/rtos/kernel/rtos.html, November 18th 2006.
14. CMX Systems. “TINY PLUS Real-Time Multi-Tasking Operating System for Microprocessors and icrocomputers” http://www.cmx.com/, November 18th 2006.
15. Barry, Richard. “FreeRTOS”, http://www.freertos.org, November 2004 – November 2006.
16. Florez, David, Rodrigez, Guillermo, Ortiz Juan. “BESA-ME: Arquitectura de sistemas multi-agente para microcontroladores”. Pontificia Universidad Javeriana. Bogotá, Colombia. 2005.
17. Gonzalez Enrique, Bustacara César, Avila Jamir,. "Agents For Concurrent Programming". CPA 2003, Enschende-Holanda, IOS PRESS, Septiembre 2003, PP 157-166. ISBN 1-58603-381-6.