

Physical Awareness and Embedded Software Agents

Dan C. Marinescu, Yongchang Ji, Gabriela M. Marinescu, Xin Bai

School of Electrical Engineering and Computer Science
University of Central Florida
Orlando, Florida, 32816, USA
[dcm, yji, magda, xbai]@cs.ucf.edu

1. EMBEDDED SOFTWARE AGENTS

Embedded systems are pervasive; according to a market study 98 % of the 4.4 billion microprocessors and micro-control units produced in 1997 were used for embedded applications. Advances in sensor technologies, low-power devices, and wireless communication will most likely reinforce this trend.

Embedded systems are generally viewed as *reactive systems*, they are expected to monitor changes in the environment using sensors and to react to a predefined set of events. According to Harell [4] a reactive system is characterized by:

- (i) asynchronous input and outputs, that change values unpredictably;
- (ii) interacting processes that operate concurrently;
- (iii) stringent timing requirements;
- (iv) preemptive behavior, the ability to respond to high priority interrupts;
- (v) diversity of operational scenarios, depending upon the current state, the inputs, and the past history.

Embedded systems have limited resources, their power dissipation is constrained, and often their reactions are subject to hard deadlines.

Embedded software agents (ESAs) are an artifact of the need to accommodate increasingly more complex embedded systems and to exhibit some degree of autonomy and mobility. The complexity of embedded systems results from multi-modal sensory perception and a large palette of actions the ESA are expected to choose from.

The *autonomy* of systems capable of *physical mobility* guarantees that an embedded system out of touch with a controlling entity for prolonged periods of time is still capable to attain its objectives. In addition to physical mobility embedded systems should support *virtual mobility*, the ability to move code, data, and state from one platform to another. Virtual mobility allows us to upgrade, to extend the functionality, and to repair physically mobile embedded systems.

We can only accommodate complexity and autonomy with software systems that behave intelligently. This means that in addition to reactive behavior the embedded system should

have the ability to plan its actions, to learn from its past experience, and to infer new facts given a set of rules and facts.

Combining mobility with reactive and intelligent behavior is a non-trivial endeavor. Reactive responses are typically subject to timing constraints. The deadlines for taking an action vary widely; milliseconds for an ESA controlling the anti-skid system of a car, seconds for a target recognition system, minutes for an automatic sprinkler system, hours for some systems of a spacecraft. On the other hand, intelligent behavior involves searching very large spaces in case of planning or repeated matching operations in case of inference. Learning is by definition a process that takes a fair amount of time to accumulate knowledge. Both reactive and intelligent behavior require awareness of the environment.

The software systems running on the embedded microprocessors and on micro-control units test the limits of our computational models. Models based upon resource virtualization allow us to reason about algorithm complexity but do not consider the diversity of computational platforms, the limited resources available on each platform, the effect of environment, or the time. Abstractions such as virtual memory, virtual machine, operation counts, ignore the physical limitations of the computing systems and the characteristics of the surrounding environment. They allow us to develop software capable to run on a diversity of platforms, but make it very difficult to enforce any constraints regarding the execution time on any given system, to adapt to a particular environment, or to self impose limitations on the amount of resources used.

For example, to plan the detailed flying path of a long-range, unmanned surveillance aircraft, the resident ESA is expected to gather input from the global positioning system (GPS), from sensors monitoring the wind force and the amount of fuel in the tanks and from visual images of the terrain the aircraft overflies. The decision whether to transmit a stream of images or to run more sophisticated target identification programs is driven by the limited amount of power available to the computing and communication systems installed on the aircraft. At times, the aircraft may either be out of range of the controllers or may need to be silent to avoid detection, thus the need for autonomy. We may need to send to the aircraft a repair agent able to cope with unexpected conditions e.g., a solar eruption affecting communications with the GPS system or a strong magnetic field perturbing the navigation system.

2. PHYSICAL AWARENESS AND REACTIVITY OF ESAS

The term *physical awareness* refers to the ability of an ESA to gather information about its environment. There are several aspects of physical awareness: *Sensory awareness* refers to the ability of an agent to gather visual images, to register sounds, to measure the physical, thermal, and electrical characteristics of the environment; *Spatial* and *temporal* awareness are related to the ability to determine the location and the time; *Computing environment awareness* is related to virtual mobility. A mobile agent may need to know the architecture or the type of microprocessor, the hardware configuration, the operating system, the amount of main memory and secondary storage, the speed of communication links, the residual amount of power available the computing environment on a target platform [5].

A possible approach to accommodate physical awareness, reactivity, and intelligent behavior is to assemble dynamically an agent from pre-existing components. The notion *well-structured* refers to agent with a structure based upon finite state machines. In addition, it requires the ability to load dynamically the functional components before the system enters a state. The structure itself may consist of multiple threads of control and may be altered dynamically to enhance or to adapt the functionality of the agent. An agent description language is used to describe the structure of an agent. Some of the functional components are responsible for the reactive behavior, others are responsible for the intelligent behavior including planning, learning, and inference.

A component-based agent architecture based upon the ideas outlined above is dissected in Chapter 8 of [7]. The structural component of an agent, the *blueprint*, and the functional components, *strategies*, come from local or from the remote repository. The agent has multiple planes, each plane is a state machine, [1, 2]. Each state of a state machine has a strategy associated with it. Once created, the *action scheduler* and the *semantic engine* determine the flow of control of all agent actions using an internal data structure. Strategies can be loaded dynamically from local repositories (*S2*), from Web servers (*S1*), or may be written in a scripting language and embedded in a message from another agent (*S3*). Strategies communicate with one another through the model or using a tuple space [8]. Different planes typically capture different aspects of agent behavior, for example, one plane may be responsible for interactions with the controlling entity, another one for monitoring temperature sensors, a separate plane may collect the facts and run the inference engine, and so on. Typically individual planes run on separate threads of control.

The *agent factory* translates a *blueprint* into an internal control structure and an agent. When a *surgical blueprint* is provided, the agent factory modifies the internal data structure controlling the agent and is able to automatically generate the modified blueprint. Several agent transformations are defined including *split*, *join*, *trim*, and *augment*; the first two are operations on planes the last two modify the states in a plane. Splitting an agent involves sending one or more planes of an agent to an existing agent or creation of a new agent at a different location. Join is the reverse of a split. Trimming involves removing states that cannot be reached from the current state while augmentation adds new states

to a plane.

The system described above is based upon *weak mobility*, has facilities for fault detection, supports interoperability with T Spaces and Jini, has an inference strategy based upon an inference engine developed at Sandia National Laboratory [3], yet it was not specifically designed for embedded applications. To fully support the reactive behavior and the physical awareness the architecture outlined above needs several extensions. First, we need to structure some of the planes and strategies using a high-level model and describe them in a language suitable for reactive systems, for example using the Statecharts model [4]. Second, individual components should have a descriptor that defines the function performed, as well as the computing resources needed to perform the function. Identification of the optimal component is a process of matching the attributes of the target platform, **TPattributes** with the attributes of the component, **Cattributes**. To avoid long delays for searching for the optimal component when the system enters a given state, the dynamic loader may use a look-ahead planner to identify the optimal components before the corresponding state is reached.

3. ACKNOWLEDGEMENTS

The research reported in this paper was partially supported by National Science Foundation grants MCB9527131, DBI0296107, ACI0296035, and EIA0296179.

4. REFERENCES

- [1] L. Bölöni, K. Jun, K. Palacz, R. Sion, and D. C. Marinescu. The Bond Agent System and Applications. In *Proc. 2nd Int. Symp. on Agent Systems and Applications and 4th Int. Symp. on Mobile Agents (ASA/MA 2000)*. Lecture Notes in Computer Science, volume 1882, pages 99–112. Springer-Verlag, Heidelberg, 2000.
- [2] L. Bölöni and D. C. Marinescu. A Multi-plane Agent Model. In *Autonomous Agents, Agents 2000*, pages 80–81. ACM Press, New York, 2000.
- [3] E. Friedman-Hill. Jess, the Java Expert System Shell. Technical Report SAND98-8206, Sandia National Laboratories, 1999.
- [4] D. Harel and M. Politi. Modeling Reactive Systems with Statecharts: the Statemate Approach. ISBN 0-07-026205-5, Mc. Graw Hill, New York, 1998.
- [5] K. Jun, L. Bölöni, K. Palacz, and D. C. Marinescu. Agent-Based Resource Discovery. In *Proc. Heterogeneous Computing Workshop 2000*, pages 43–52. IEEE Press, Piscataway, N.J., 2000.
- [6] D. C. Marinescu. Reflections on Qualitative Attributes of Mobile Agents for Computational, Data, and Service Grids. *Proc. 1st IEEE/ACM Int. Symp. on Cluster Computing and the Grid 2001*, pages 442–449, IEEE Press, Piscataway, New Jersey, 2001.
- [7] D. C. Marinescu. Internet-Based Workflow Management: Towards a Semantic Web, ISBN 0-471-43962-2, Wiley, New York, 2002.
- [8] L. Tobin, M. Steve, and W. Peter. T Spaces: The Next Wave. *IBM System Journal*, 37(3):454–474, 1998.