An Agent-based Architecture for Resource-Aware Mobile Computing

Yun Ding and Rainer Malaka
European Media Laboratory
Villa Bosch, D-69118 Heidelberg, Germany
Yun.Ding@EML.Villa-Bosch.de
Rainer.Malaka@EML.Villa-Bosch.de

Keywords
Adaptive assistance architecture, resource-awareness, agent programming environment, quality of service, load balancing

Abstract
Mobile computing is one of the major trends in information technology and goes far beyond carrying laptops from the office to the home desk. Current trends in mobile computing can be summarized by the new paradigm of disappearing computers. This new metaphor aims at integrating computers into the user’s environment in such a way that the computer is not perceived as a technical device anymore. The computing power however does not reside in the tangible interfaces, e.g., smart glasses or earphones, but is located either in some wearable devices or in embedded computers, e.g., in a TV set or a car. In order to provide a flexible infrastructure that adapts to a dynamic setting of networks and services as it is addressed in the project framework EMBASSI, intelligent mechanisms are required that make efficient use of available resources. Here we describe a novel agent-based middleware architecture, which is able to support a wide range of adaptation strategies while taking application-specific knowledge into account. The proposed architecture allows structured programming by using a multi-level architecture.

1 Introduction
There are two trends in computer technology that dramatically change the present paradigm of personal computers (PCs). On the one hand, devices become smaller and more personal. On the other hand inexpensive computing power, storage capacity and networking capacities are being integrated into all sorts of devices and turn consumer electronic devices such as refrigerators, microwave ovens or VCRs into smart devices. The time of monolithic mainframe systems with many users is already over for some time and PCs were the first milestone in this evolution. The PC is still getting smaller and portable computers such as powerful laptops and less powerful but lightweight personal digital assistants (PDAs) are now reaching the mass market. The next step is marked by wearable computers that are carried by the user like cloths and by new display technologies like head-mounted displays allowing new ways of augmenting the user’s reality. The preliminary end of this technological evolution seems to be the disappearing computer where computers are not perceived as technical devices anymore, but rather form a computing environment in the user’s world [16, 10]. In a large German initiative of the German Federal Ministry for Education, Science, Research and Technology (BMBF) the focus project EMBASSI brings together about 20 partners from industrial and academic research institutions working on electronic assistance in such a computing environment, namely in the private home, in cars or at information terminals. One goal of this project in which also partners from the consumer electronics actively participate is to provide an infrastructure that allows seamless integration of embedded computers in cars, TV sets, VCRs and mobile devices. Such a complex environment without predefined set of available resources requires a high degree of flexibility and adaptivity to the prevailing situation. In particular, when resource-consuming components such as live-like characters, high-quality visualizations, and natural
language processing are involved, it is crucial to find the optimum assignment of processors to processes in order to guarantee an acceptable performance and response time. The execution of a NLP module may for instance result in a long response time due to the limited computational power of a small wearable computer, making it less comfortable for potential users. In this case, one might decide to either turn down the NLP interface and employ some less complex graphical user interface or to locate the NLP system on a more powerful computer, e.g., embedded in a nearby TV set, and link the wearable with the TV-computer through some wireless network.

In general, an intelligent solution for resource-aware and adaptive computing should cover a range of adaptation strategies, e.g. resource-aware adaptation of services, context-dependent service migration, and service-oriented anytime algorithms. The problem of combining all these strategies is still an unsolved and very hard task. It is necessary to monitor resources, services, and also user preferences and contexts. In the framework of EMBASSI, which is designed to be extensible to new devices, it is also important to model services such that they come along with necessary information on their tasks, their capabilities, performance requirements, and other important factors. In the following, we outline an agent-based resource-adaptive infrastructure that supports all three strategies of service adaptation, service migration, and anytime algorithms within one system. This, in turn, can then be used to build an adaptive system where the computers disappear.

2 An Infrastructure for Resource-Adaptive Mobile Computing

It is widely accepted that in order to guarantee the best quality of service (QoS) applications have to adapt to the dynamically changing computing and communication environment [9], which is rule rather than exception in mobile computing. There exist a number of possibilities to do resource-adaptive computation. Here we distinguish between three classes of strategies:

- Resource-aware adaptation of services: In the scenario of video streaming in a mobile environment, e.g., the fidelity of a video can be adapted according to the available networking resources. When high bandwidth connections are available, full video can be transmitted, with lower bandwidth, compressions of the video or even slideshows can be transmitted to save the bandwidth and ensure a prompt response [14].

- Context-dependent service migration: this includes remote execution of application and application migration: Speech recognition, for instance, can require a great deal of computational power and response time on small mobile devices can get long. In this case, the mobile device could look for more powerful machines in reach of the wireless network and can delegate the service. Service migration minimizes the execution time of single tasks and increases the overall throughput of the system.

- Service-oriented anytime algorithms: There is always a tradeoff between response time and response quality. Anytime algorithms can be stopped at any time and guarantee at least some useful result. The response quality, however gets better the longer the algorithm is allowed to continue its execution. Information retrieval is a good example where a smart system could take user preferences into account and thus provide the patient user a tailored answer, while it provides at least a useful answer when impatiently interrupted.

The applicability of these strategies depends on the particular type of service. We introduce RAJA, a Resource-Aware Java Agent Infrastructure, which supports all the three adaptation strategies outlined above and can be easily extended to support more elaborate strategies.

RAJA is a two-level architecture (Fig. 1):

- At the application-level basis agents model domain-specific functionalities.
- At the system-level meta agents provide infrastructure services such as monitoring and controlling of resources, load balancing, remote execution and scheduling of multi-agent activities.

Additionally, controllers are attached to resource-adaptive basis agents to support customizable adaptation for each basis agent, while a resource-unaware basis agent does not have a controller. The following will describe the RAJA components in more detail.
2.1 Meta agents
Existing specifications promoting interoperability of agents such as KQML [3] and FIPA [5,6] define a set of system agents to facilitate agent management, agent communication and “yellow-pages” directory services, with which agents register their services and query for services offered by other agents. With the exception of FIPA 2000 [6], which is concerned with providing an infrastructure to monitor and control the ever-changing communication connections, existing specifications do not address the aspect of resource-adaptivity. RAJA is able to support adaptivity for different kinds of resources. They range from physical resources such as CPU performance and memory or bandwidth capability, user’s resources like time constraints, working-memory load and emotion, to situational/contextual resources determined by the users’ location and their social environment.

To address resource-adaptivity RAJA provides a set of predefined meta agents, which monitor the available resources (Resource Monitor) and help basis agents to adapt themselves according to the prevailing resource situation. For example, the meta agent Load Balancer collects overall load information and decides where to perform a pending task. Remote Execution executes the decision of the Load Balancer by transferring the pending task to the actual remote executor. After decomposing a complex request from the user into a sequence of sub-requests, the meta agent Scheduler determines the order of the sub-requests in order to exploit parallelism. RAJA also provides tools for agent developers to design and implement their own meta agents.

2.2 Basis agents and controllers
Although the meta agents form the basis of the infrastructure, additional requirements must be fulfilled to implement adaptive applications in a real system. These are:

- **Tailored adaptation:** Application-level resource management which takes domain-specific knowledge into account, is the key to an individually tailored adaptation for each basis agent.
- **Efficiency:** In order to avoid a communication bottleneck the amount of interactions between basis and meta agents should be kept as low as necessary. Moreover, the overhead for computing the adaptation strategy, e.g., if another set of parameters or configuration (migration or not) should be used, and the adaptation itself should be kept small such that the agents’ resources are not significantly reduced.
- **Support for resource-unaware legacy software:** To benefit from software re-use it should be possible to convert resource-unaware legacy software into resource-aware software with the help of the proposed infrastructure.
To meet the requirements we suggest attaching an individual controller to each resource-aware basis agent. A controller is comparable with the personal secretary of a basis agent. It

- manages domain-specific knowledge such as the dependency between application parameters (e.g. if the noise level is high, the recording quality of speech is low), and the relationship between application parameters and resource consumption (e.g. speech recognition with noisy speech consumes more CPU time than with studio speech),
- contacts the meta agents to be aware of changes in the resource availability,
- works out an application-specific adaptation strategy for its basis agent.

Basis agents and their controllers interact through method calls. Whenever asked by a basis agent to make an adaptation decision, its attached controller contacts some meta agents only if the decision can not be made based on local information. Since basis agents and controllers run in separate threads, adaptation strategies can be calculated asynchronously to the domain-specific computation.

The distinction between basis agents and their controllers clearly separates application-level resource management undertaken by a controller from domain-specific computation, for which a (resource-unaware legacy) basis agent is responsible. This separation eases the transformation of a resource-unaware application into a resource-aware application.

Basis agents and meta agents communicate with each other via message passing. Meta agents are not allowed to access and modify the state of basis agents directly as in the TLAM-approach [15]. Instead, controllers interact with meta agents on behalf of their basis agents and thus ensure their autonomy. Each interaction between a basis and a meta agent follows a particular interaction protocol (e.g., FIPA-request- or FIPA-query-protocol [5]), which is determined by the particular service provided by the meta agent.

The interactions between a basis agent, its controller and meta agents are briefly illustrated in Fig. 2: At first the basis agent passes application-specific knowledge such as QoS-requirements and relationships between parameters and resource consumption to its controller (step 1). The controller starts with the help of the meta agent Profiling Service the application profiling (step 2). Based on the profiles the controller calculates the values of...
substantial resource changes, at which an adaptation should be triggered, and subscribes to the notification of changes of resource availability at the Resource Monitor (step 3). The controller computes its application-specific adaptation strategy after a notification (step 4) and being supplied with system-level adaptation strategies (step 5). The strategy is made visible to the basis agent that has to perform the actual adaptation (step 6). The controller subsequently updates the application profiles (step 7). Note that the controller runs asynchronously to the basis agent, so it does not prevent the basis agent from executing the application functionality except during the adaptation itself.

3 Implementation
The system agents defined by the FIPA specifications enable the agents to be interoperable, while our meta agents give the basis agents the possibility to be adaptive in a continuously changing environment. To combine interoperability with adaptivity and to save programming cost we decided to implement our RAJA infrastructure on top of an existing “interoperable agent infrastructure”. After evaluating several of them we chose the FIPA-compliant JADE Framework [1]. The API of RAJA exploits only objects specified by the FIPA specifications using their original names, and provides all elementary mechanisms specified for agent management as well as for agent communication. Thus, RAJA can easily be ported to other agent infrastructures such as FIPA-OS [4] without affecting the agents implemented based on it.

The RAJA agent infrastructure defines a set of Java classes, which can be extended to develop resource-adaptive agents. Java classes BasisAgent, Controller and MetaAgent are available for the components basis agents, their controllers and meta agents respectively:

- The class BasisAgent offers a number of methods for agents to utilize the services provided by the FIPA system agents. Additionally, it provides methods for agents to specify their task structure. This knowledge is required to schedule sub-tasks efficiently at runtime.
- The class MetaAgent provides methods for meta agents to specify their services, which are called MetaAgentsAction. For instance, Resource Monitor handles queries to resource availability and Load Balancer accepts requests to update the load information. An agent developer has to implement the actions undertaken by the meta agents to provide their services.
  - Each method call from a basis agent, which cannot be handled by its controller locally, must be transformed into messages to some meta agents. Upon receiving replies from them, the controller makes an adaptation decision. The class Controller provides methods to ease the transformation from method calls to messages as well as to specify the response actions performed by a controller after getting replies from some meta agents.

In RAJA all the three classes are equipped with a dispatcher, which surveys the message traffic. For basis agents, it puts incoming messages into buffers. For meta agents and controllers, after unpacking an incoming request or a reply message, it invokes the action, which must be performed by a meta agent to provide the requested service or by a controller to operate on the reply. Message handling performed by the dispatchers is completely hidden from agent developers, which substantially eases the agent development.

4 Related Work
Since the introduction of standardization approaches for interoperable multi-agent systems a number of agent architectures (e.g. KQML-compliant Jackal [2] and JATLite [8] or FIPA-compliant JADE [1] and FIPA-OS [4]) have been published. Like the existing specifications they do not address resource-adaptivity.

In the context of pervasive and mobile computing numerous middleware for resource-adaptive applications have been developed. Examples are the non-agent based approaches such as Odyssey [12, 13] and Agilos [9], and the actor- or agent-based approaches like TLAM [15] and DECAF [7]. RAJA fills the gap of supporting resource-adaptivity left by existing specifications. It distinguishes itself by its greater flexibility, which characterizes agent-based approaches and by its concept of controllers, whose advantages have been demonstrated in detail in section 2.2.
5 Conclusion and Future Work

The RAJA infrastructure presented in this paper addresses the problem of optimized resource usage in a distributed and flexible network of computers. Even though we are particularly interested in home environment scenarios as in EMBASSI where the single computational resources can be embedded into home entertainment devices, the infrastructure is more general and can be of use in a range of different scenarios where agent-based services are to be distributed over a network of computers. The main advantage of the agent-based approach with basis agents that are linked to meta agents via controllers allows a very flexible integration into existing systems as long as they follow the agent paradigm. Currently our implementation of RAJA supports JADE and can easily extended to support other FIPA platforms such as FIPA-OS. Even a heterogeneous system that uses different agent platforms is possible.

It is, of course, not possible to provide application specific, resource-aware mechanisms for QoS optimization. This has to be done by the agent programmers at the level of the basis agents and their controllers. The meta agents, however, do all the system-level resource management and the programmers do not have to take care for these aspects.

We support the three most important resource-aware adaptation strategies, resource-aware adaptation of services, context-dependent service migration, and service-oriented anytime algorithms. For each basis agent, one or more of these strategies can be integrated into the agent’s controller and thus turns the agent into a resource-aware agent.

If a whole existing system has to be transferred into a resource-aware system, one can start with the most critical agents turning them into resource-aware agents and can leave the others as they are. This makes a gradual migration feasible and allows to re-use existing agent systems. Currently, we begin to test this approach in the Deep Map framework, a complex agent system [11]. In further steps we will integrate this technology into EMBASSI allowing agents to adapt to networking and computing infrastructures of private users and their home entertainment infrastructure.

Acknowledgements

The authors thank their colleagues at EML for fruitful discussions and comments on the manuscript. The work presented here has been funded by the Klaus Tschira Foundation and the German Federal Department for Education, Science, Research and Technology (BMBF), FKZ 01 IL 904 D2.

References


