Engineering Ambient Intelligence Systems Using Agent Technology

Nikolaos Spanoudakis, *Technical University of Crete*

Pavlos Moraitis, *Laboratory of Informatics Paris Descartes (LIPADE)*

The HERA project, run by a consortium of researchers from academia and industry, applied the agent systems engineering methodology (ASEME) to develop a multi-agent system for a real-world application for ambient assisted living.

Engineering an ambient assisted living (AAL) information system using agent technology is a nontrivial task.[1] AAL systems combine the application domains of ambient intelligence (AmI) and home-based e-health. AmI refers to environments empowered with computing capability that deliver services and applications to people: autonomy, distribution, adaptation, and responsiveness are key characteristics of AmI computing components.[2]

Researchers working on the Home sERvices for specialised elderly Assisted living (HERA) project (w3.mi.parisdescartes.fr/hera), supported by the AAL Joint Programme (www.aal-europe.eu), built an AAL system to provide cost-effective specialized assisted living services for elderly people suffering from mild cognitive impairment (MCI), mild/moderate Alzheimer's disease (AD), and other diseases (diabetes, cardiovascular) with identified risk factors, aiming to significantly improve the quality of their home life and extend its duration. To address these needs, HERA provided the following main categories of services:

- cognitive exercises,
- passive communication,
- pill and exercise reminders,
- reality orientation (date and time), and
- blood pressure and weight monitoring.

In HERA, we used a combined agent- and service-oriented software development approach. A service-oriented architecture based on Web services allowed the different subsystems to be connected in a plug-and-play standardized way, while agent technology allowed for service personalization. The use of the TV set and remote control for human-machine interaction allowed for a quick learning curve for our users.

In this article, we describe how we built the intelligent part of the HERA system, using an agent-oriented software engineering (AOSE) methodology, specifically the agent systems engineering methodology (ASEME; www.amcl.tuc.gr/aseme). AOSE is concerned with defining the metaphors, concepts, and methods involved in agent-based systems, which, because of the agents themselves, are proactive (have goals and pursue them, take the iniative to act for

1541-1672/15/$31.00 © 2015 IEEE
Published by the IEEE Computer Society

the achievement of their goals), reactive (respond to events), social (acquainted with other similar software and can cooperate-compete with it), autonomous (don't require human intervention), and intelligent (may perform tasks that require a certain intelligence).

These characteristics motivated us in using agent technology for the HERA project.

Development phase	Levels of abstraction		
	Society level	Agent level	Capability level
Requirements analysis	Actors	Goals	Requirements
Analysis	Roles and protocols	Capabilities	Functionality
Design	Society control	Agent control	Components
Implementation	Platform management code	Agent code	Capabilities code

Figure 1. The agent systems engineering methodology (ASEME), phases, and abstraction layers. The figure shows which agent modeling elements are better suited for each development phase and abstraction layer.

ASEME

Agents are computational entities that have several capabilities, such as sensing, decision making, planning, learning, and so on. In the context of multi-agent systems, several agents can interact by using different kinds of protocols, which implement various types of dialogues (such as negotiation, deliberation, and persuasion). These protocols form what we call inter-agent control.

AOSE methodologies are concerned with the development of intelligent agents and multi-agent systems. ASEME combines successful concepts from the AOSE domain with the model-driven engineering (MDE) paradigm. In MDE, models guide the development process and are systematically transformed to system implementations. In this approach, the models of each software development phase (analysis, design, or implementation) are produced by applying transformation rules to the models of the previous phase. Successive models add more detail and become more formal, gradually leading to full implementation.

ASEME, uniquely among AOSE methodologies, considers integration of the inter-agent and the intra-agent control models (EAC and IAC, respectively) by using a uniform representation (that is, statecharts). The first defines the protocols that govern agent coordination, whereas the latter defines agent behavior by coordinating the different modules that implement agent capabilities. ASEME uses the Agent Modeling Language (AMOLA), which provides the syntax and semantics for creating models of multi-agent systems.

Another novelty of ASEME is the use of three levels of abstraction in each development phase. The first is the *societal level*, where the whole multi-agent system functionality is modeled. The *agent level* zooms in on each part of that society (that is, the agents themselves). Finally, the *capability level* defines each agent's elementary components. Figure 1 presents the ASEME phases, the different levels of abstraction, and the AMOLA concepts related to each.

In the following, we show that agent technology (and ASEME, in particular) is useful for modeling real-world AmI systems, where intelligence, autonomy, and proactiveness are necessary features. The system validation process that we followed in HERA helped identify and measure our approach's added value.

HERA Software Engineering Process

The HERA project used the ASEME process in a step-by-step manner to develop a multi-agent system (MAS) module. The process was incremental—the HERA system's deployment took place in two major stages, first in a medical center for an initial evaluation with medical personnel and for controlled interaction with patients and then inside users' homes for a final evaluation. The primary user groups for the HERA system were patients (elderly people over the age of 65, suffering from mild AD, moderate AD, or MCI) and medical personnel.

The HERA project started with a five-month requirements analysis and specification work package where the system architecture, subsystems identification, and requirements cataloguing took place. Then, each subsystem was developed in a separate work-package that lasted 14 months. A first version of the subsystems was released after the first eight months and a second version at the end of the work-package.

The first deliverable of the work-package was the Agent Interaction Protocols, which was released after the fifth month. It identified the agent types to be implemented, documented the MAS subsystem analysis, and defined the interactions between the agents (inter-agent control) and the interfaces with the other HERA subsystems or modules. The design of the individual agents (intra-agent control) and their capabilities was completed in the respective agent development tasks and was released with the software.

Requirements Analysis Phase

To determine requirements, AMOLA defines the system actors and goals (SAG) model, which is similar to the Tropos actor diagram,[3] with actors

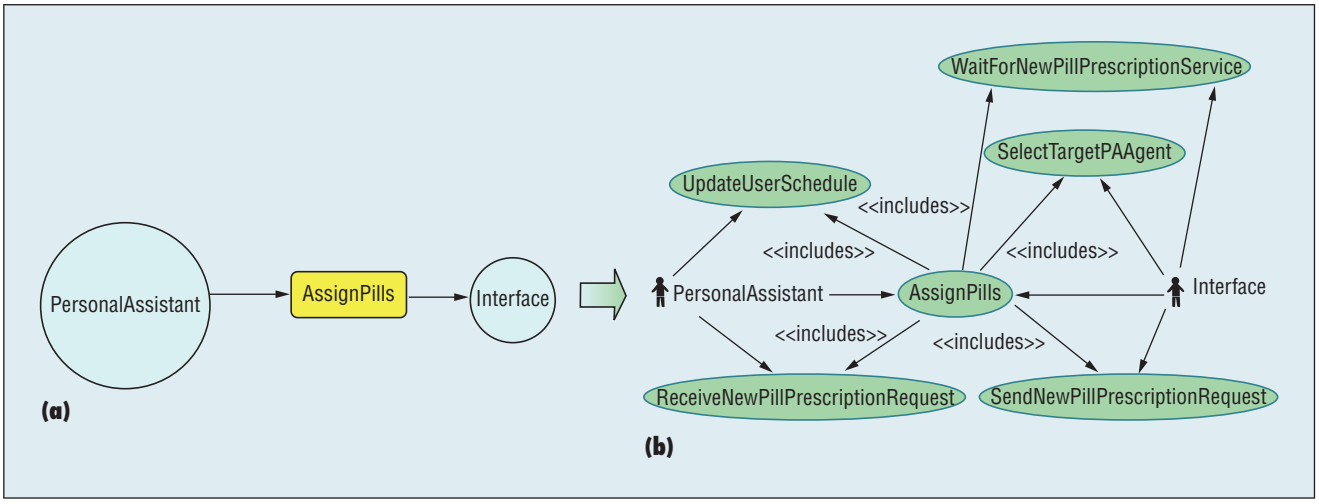

Figure 2. A portion of the (a) system actors and goal (SAG) model and (b) the system use cases (SUC) model. In the SUC diagram, the AssignPills goal has been transformed to the Assign pills use case. During the analysis phase, using top-down decomposition, this general use case was decomposed to several more specific ones, using the «include» relationship for both participating roles. The personal assistant, for example, will need to receive a new pill prescription request and update the user's schedule to achieve the assign pills use case.

and their goals (requirements are associated with goals in free, unstructured text).

In HERA, we identified a personal assistant (PA) actor that serves a specific user and has access to that user's profile data. This actor is proactive in the sense that it's always actively following the user's agenda and providing support and advice by taking action whenever needed. For example, it can send a message to the user to remind him to take his pills or to receive sensor information from the user. The PA also uses request history to adapt its services to the user's habitual pattern. We also had an interface actor to serve as a gateway to the PAs. Thus, while the PA itself accesses all the information sources it needs, the HERA back office could send additional information to the MAS through the interface actor—for example, the PA's AssignPills goal depends on the Interface (see Figure 2a), thus there's a line from the depender (PA) to the goal and then from the goal to the dependee (Interface). For reference, the back office is an actor representing the HERA server stakeholder who holds the information coming from the various other stakeholders—for example, the HERA server [or back office] holds information on new pill prescriptions from the medical personnel stakeholder. For this information, the PA depends on the Interface actor, who knows all personal assistants and can match the information.

Analysis Phase

As with any analysis phase, after identifying requirements, it's time to transform them into technical needs. To this end, AMOLA defines the system use cases model (SUC) to decompose goals into generic activities, the Agent Interaction Protocol (AIP) model to define the protocols that govern inter-agent interactions, and the system roles model (SRM), inspired from the Gaia methodology,[4] to define how a single role realizes zero or more interaction protocols.

The first task of this phase is to define the SUC model, which is initialized by the requirements phase's SAG model. Specifically, SAG actors are transformed to SUC roles (depicted as Unified Modeling Language [UML] use case diagrams' roles) and SAG goals to SUC use cases (depicted as UML use case diagrams' use cases). The analyst refines the SUC model by decomposing high-level tasks to simpler activities by using the familiar UML «include» relationship. Figure 2b shows the automated transformation process from the SAG model to a SUC model with the AssignPills use case, which has been decomposed into several other use cases (using the «includes» relationship), three of which are connected to the Interface role and the other two to the PersonalAssistant role.

Next, the analyst constructs the AIP model, which focuses on the different roles' joint use cases—all use cases with more than one participant define an AIP model instance. Each protocol defines the participating roles, the rules for engaging, expected outcomes, and the process that each participant follows within the protocol. Figure 3a shows the AssignPills use case modeled with AIP; the yellow box shows the protocol name and is connected to as many pink boxes as there are participants. Each pink box

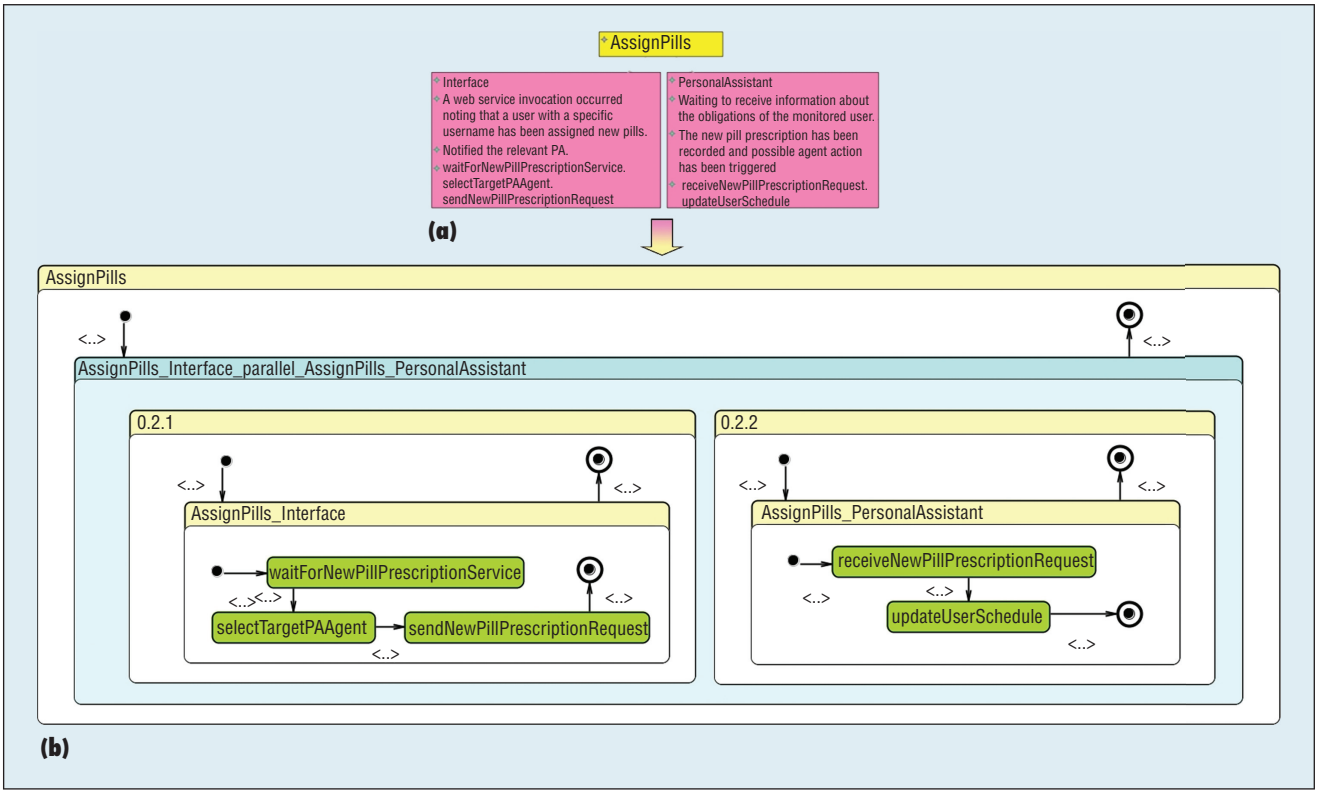

Figure 3. From the analysis to the design phase in the societal level of abstraction. (a) The AssignPills Agent Interaction Protocol (AIP) model, where the yellow box shows the protocol name and is connected to as many pink boxes as there are participants, and (b) its transformation to an inter-agent control model, where the process part of the AIP has been transformed to a statechart.

contains four bullets with (from top to bottom):

- the participant role's name;
- the preconditions that must hold so that the participant role can enter the protocol (in free text format);
- the postconditions that must hold when the participant role exits the protocol (in free text format); and
- an expression describing the process followed by the participant role within the protocol.

The process is defined formally using liveness operators to connect activities. Briefly, A.B means that activity B is executed after activity A executes, A* means that A is executed zero or more times, A+ means that A is executed one or more times, A~ means that A is executed indefinitely (it resumes as soon as it finishes; the equivalent to the ω operator in the original Gaia formalism), A|B means that either A or B is executed exclusively, A||B means that A and B are executed concurrently, and [A] means that A may or may not be executed.

The AssignPills protocol has two participant roles (Interface and PersonalAssistant). Taking a closer look at their process fields, we can see the following actions:

- The Interface waits for a new pill assignment to be submitted (using the WaitForNewPill PrescriptionService activity). When the service is invoked, it determines the relevant PA (using the SelectTargetPAAgent activity) and sends an appropriate request message (using the SendNewPillPrescription-Request activity).
- The PersonalAssistant waits to receive such a message (using the ReceiveNewPillPrescriptionRequest activity) and updates its user's schedule accordingly (using the UpdateUserSchedule activity).

Subsequently, the SUC and AIP models are transformed to SRM models. We build one SRM for each role that will be developed as an agent (the analyst's choice). In the SRM we use liveness formulas[4] for defining the agent capabilities and connecting them with each other to define the overall role's process. Liveness formulas have a name on their left-hand side (representing a capability) and an expression built using the liveness operators on the right-hand side.

Figure 4 presents the SRM model for the Personal Assistant (PA) role. It includes the role's name, capabilities, activities, and the liveness formulas. The top formula has the role name on the left-hand side and an expression on the right-hand side. Thus, the PA role process is the following: first, it executes the PersistRegisterBehavior activity, optionally followed by the PersistLoaderBehavior activity. Then the StoreAgentState, AssignExercises, AssignPills,

Role: Personal Assistant
Capabilities: StoreAgentState, UserSwitchedOnTV, UpdateUserSchedule, ResolveConflicts, UpdateUserProfile, RemindUser, RemindUserOfTask, GetFeedbackOnUserAction, LearnUserHabits, AssignPills, AssignExercises
Activities: PersistRegisterBehavior, PersistLoaderBehavior, WaitForMidnight, AutoSaveBehavior, ReceiveUserSwitchedOnTVInform, UpdateUserProfileStructure, InvokeAvailableVideosInformService, UpdateUserScheduleStructure, CheckIfConflictsExist, ReasonOnItemsPriorities, SortItems, ReceiveUserProfileUpdateRequest, WaitForUserScheduleNextItem, InvokeBloodPressureModule, ReasonOnPillsQuantity, InvokeNotificationModule, ReceiveUserActionInform, SelectNewDate, ReasonOnUserAction, SendNewPillPrescriptionRequest, ReceiveNewExercisePrescriptionRequest
Liveness:
PersonalAssistant = PersistRegisterBehavior. [PersistLoaderBehavior]. (StoreAgentState~ || AssignExercises ~ || AssignPills ~ || UserSwitchedOnTV~ || RemindUser~ || UpdateUserProfile~)
StoreAgentState = WaitForMidnight. AutoSaveBehavior
UserSwitchedOnTV = ReceiveUserSwitchedOnTVInform. UpdateUserProfileStructure. [InvokeAvailableVideosInformService. UpdateUserSchedule]
UpdateUserSchedule = ResolveConflicts. UpdateUserScheduleStructure
ResolveConflicts = (CheckIfConflictsExist. ReasonOnItemsPriorities. SortItems) +
UpdateUserProfile = ReceiveUserProfileUpdateRequest. UpdateUserProfileStructure
RemindUser = WaitForUserScheduleNextItem. [InvokeBloodPressureModule. ReasonOnPillsQuantity]. RemindUserOfTask
RemindUserOfTask = InvokeNotificationModule. GetFeedbackOnUserAction. LearnUserHabits
GetFeedbackOnUserAction = ReceiveUserActionInform. [SelectNewDate. UpdateUserSchedule]
LearnUserHabits = ReasonOnUserAction. [UpdateUserSchedule]
AssignPills = ReceiveNewPillPrescriptionRequest. UpdateUserSchedule
AssignExercises = ReceiveNewExercisePrescriptionRequest. UpdateUserSchedule

Figure 4. The Personal Assistant (PA) role in the system roles model (SRM). The liveness property contains the formulas.

UserSwitchedOnTV, and, UpdateUserProfile capabilities are executed in parallel, and if they finish they're restarted. In the next formulas, these capabilities are further refined.

Having defined the SRM for the PA and the Interface roles, the next task in the ASEME process is to define each activity's functionality—this functionality is the single tool, algorithm, or method that will realize the activity. If this isn't possible, the activities need to be further decomposed so that each corresponds to a single functionality.

Design Phase

After analysis, we move to the design phase, where AMOLA defines the EAC and IAC models, which are based in statechart language.[5] The model associated with this phase's societal level of abstraction is the EAC, which defines interaction protocols. IAC, at the agent level, defines the individual agent capabilities and their appropriate interactions. Finally, in the capability level of abstraction, the designer defines each capability's functionalities. AMOLA defines the concept of capability as the ability to fulfill specific tasks (such as the task to decide in which restaurant to have dinner this evening) that requires the use of one or more functionalities. The latter refer to the technical solutions to a given class of tasks. Capabilities are decomposed to simple activities, each corresponding to exactly one functionality.

We initialize the EAC and IAC models by transforming the AIP and SRM models respectively—specifically, the liveness2statechart ASEME transformation tool uses a recursive algorithm that takes as input a set of liveness formulas and applies operator transformation templates to build the statechart. Then, the designer edits the statechart's variables and transition expressions.

Statecharts are formal models that describe complex processes and control structures using directed graphs with nodes (states) and edges (transitions); they're edited using the Kouretes Statecharts Editor tool (KSE; www.kouretes.gr/KSE). The statechart language allows six types of states: or-states (drawn as yellow rounded rectangles), indicating complex states with mutually exclusive substates (only one substate is executed each time); start-states (bold black dots), indicating the entry of execution in an or-state; end-states (circled black dots), indicating the end of execution of an or-state; and-states (blue rounded rectangles), indicating states with substates of type or that are executed concurrently; basic-states (green rounded rectangles), indicating the execution of base activities; and condition-states (circled "c"), providing the ability to make conditional transitions. Each transition from one state (source) to another (target) has an expression whose syntax follows the pattern $e[c]/a$, where e is the event triggering the transition, c is the condition that needs to be satisfied for the transition to take place when e occurs, and a is an action executed when the transition is taken. All the elements of an expression are optional.

The two participating roles in the EAC model in Figure 3b are transformed to the statechart's orthogonal components (the or-states included in the and-state): activities connected with the "." operator are transformed to states where one follows the other with a start-state before the first and an end-state after the last. The designer can now edit the transition expressions to complete each protocol's EAC model. The items that the designer can use to define state transition expressions are

the message performatives (the types of the messages), the ontology for defining message content, and timers.[6]

At this point in HERA, we defined both the Web services that the MAS would expose and that it would need to invoke, thus connecting the MAS module to the general service-oriented architecture. The next step was to transform the formulas in the SRM models to the IAC models. Once again, the statechart instantiation was automatic, and the modeler defined the transition expressions not already defined in the EAC model.

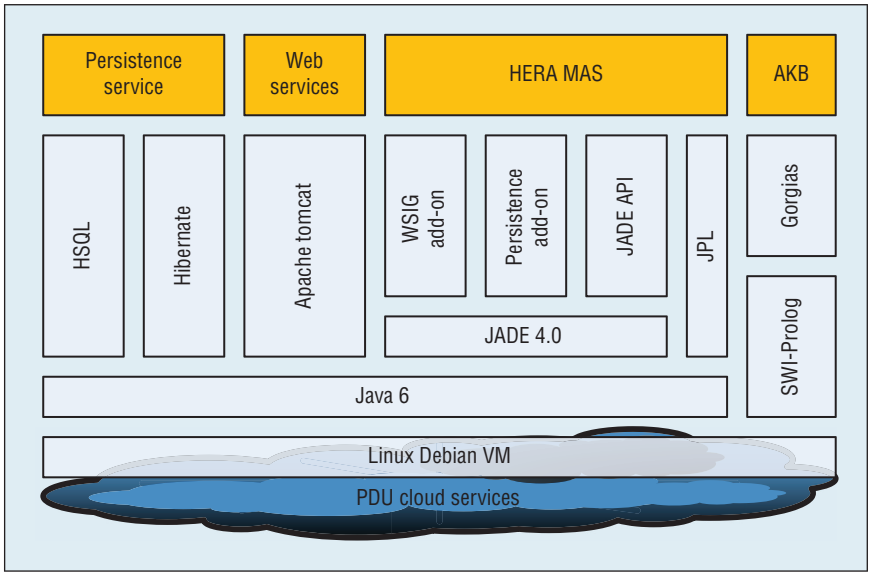

Figure 5. Server deployment scheme. AKB = argumentation-based knowledge bases; HERA MAS = HERA multi-agent system; HSQL = HyperSQL DataBase; WSIG = Web Service Integration Gateway. All the components were installed on a Linux virtual machine (VM).

Implementation Phase

In the implementation phase, the ASEME code-generation tool for the popular Java Agent Development Framework (JADE; http://jade.tilab.com) transforms the IAC models into source code. This transformation is automatic, and programmers don't have to worry about defining agents or scheduling tasks.[6] The designers are mainly focused on implementing the code corresponding to the basic-states—that is, connecting them to functionalities using specific technologies.

Figure 5 shows the software architecture for the HERA MAS module. It was deployed on a Linux virtual machine in the Paris Descartes University (PDU) cloud and used several technologies, including the Apache Tomcat (http://tomcat.apache.org) Web server for deploying the HERA agent platform's Web services. The WSIG (Web Service Integration Gateway) JADE add-on handled receiving Web service requests from Apache and making them available to agents in the form of an Agent Communication Language (ACL) message.

A JADE add-on helped with agent state persistence (the user profile and data recovery in case of system crashes) through the HSQL (HyperSQL DataBase; www.hsqldb.org) relational database and Hibernate (a tool that facilitates the storage and retrieval of Java domain objects via object/relational mapping), which we used to automate the transformation of agents' data structures to relational database tables.

Finally, we developed the HERA argumentation-based knowledge bases (AKB) that we used for reasoning on context-based situations[7] and also for dealing with conflicting knowledge by using the Gorgias (www2.cs.ucy.ac.cy/~nkd/gorgias) open source argumentation framework on top of SWI-Prolog free software. We also used the JPL (a set of Java classes and C functions providing an interface between Java and Prolog; www.swi-prolog.org) package to query a Prolog knowledge base.

Evaluation

To assess (with metrics) the added value that HERA provides, we performed an evaluation with help from previous work.[8]

We put HERA services related to the MAS module on trial in Greece. The pilot operation (phase one) took place at the Hygeia hospital, where we focused on two categories of users: patients who use HERA services and medical personnel who configure them and assess patient progress. We selected a total of 30 patients (10 healthy elderly, 8 suffering from MCI, 8 suffering from mild AD, and 4 suffering from moderate AD) to participate in this trial phase, along with 10 medical experts. The equipment installed included a set-top box (small PC), a TV set, and a remote control; Figure 6 shows the overall system architecture. The HERA application server is located on the service or telecom provider's premises (depending on the applied business model).

In the second phase, we installed the equipment in the patients' homes for a minimum of 15 days. We connected the MAS module to the HERA system so that we could adapt the service to each user's habits and needs. The system users evaluated it after both phases using a questionnaire to capture their satisfaction related to the following high-level criteria:

- Performance (C1) measures the system's capability to produce valid and accurate results.

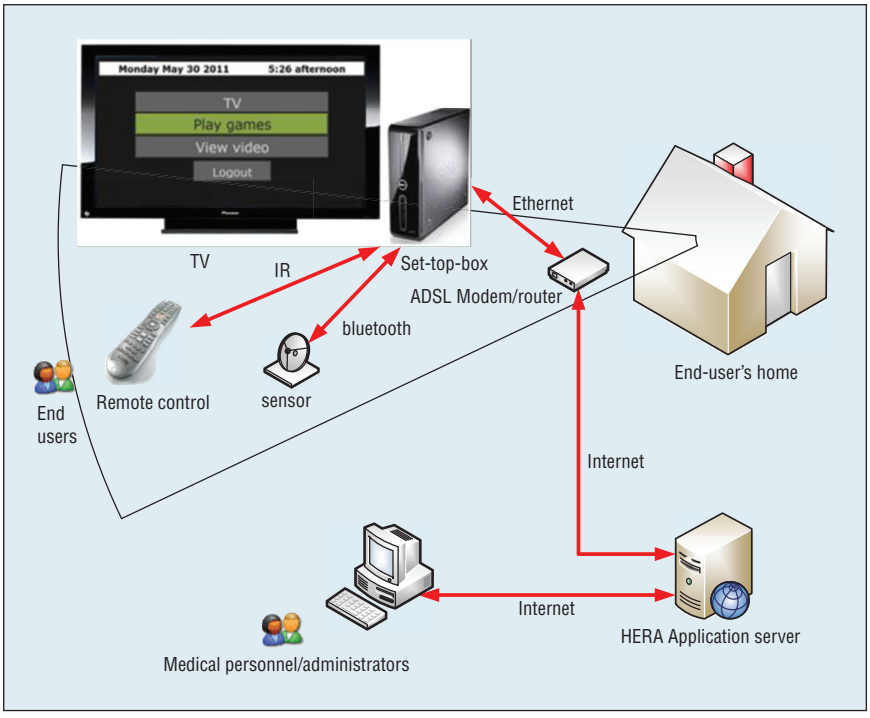

Figure 6. Overall HERA system architecture. ADSL = asymmetric digital subscriber line. This figure combines the users with the equipment, the HERA subsystems (residing on the server and set-top box) and the protocols of communication.

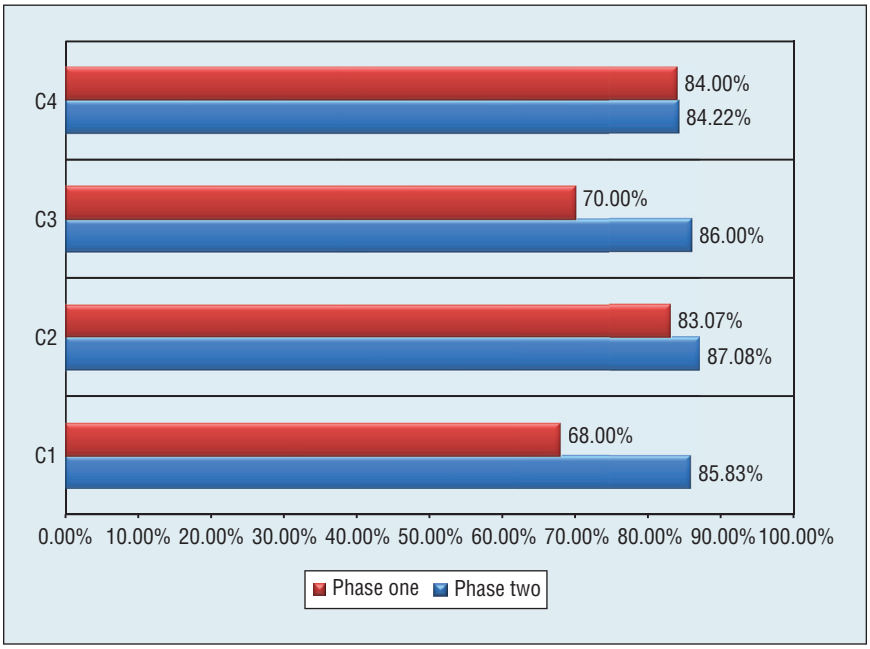

Figure 7. The performance and significance of the criteria for the patients who used HERA in phases one and two of the trials. The criteria were the system's performance (C1), usability (C2), flexibility (C3), and security and trust (C4). In the second phase, the services included personalization (the MAS subsystem) among other technical fixes.

- Usability (C2) measures satisfaction with regard to the user's experience with the system and the ease in achieving tasks with it.
- Flexibility (C3) refers to the ease of troubleshooting and of moving from one service to another.
- Security and trust (C4) measures how much the user trusts the system to securely handle data and successfully manage his or her agenda.

Figure 7 shows the system's performance for the end-user category for the first and second trial phases. The big difference in the performance criterion (C1) is mainly due to two factors:

- personalization and adaptation to user habits, as the software agents that assisted the users only became active in the second phase; and
- system improvements according to users' comments.

The Flexibility criterion (C3) was also better at home mainly because the end users overcame their initial reservation about their ability to use HERA at home: the wider the extent of cognitive impairment, the lower the perceived ability to use the system without immediate assistance.

This article showed how a practitioner can apply the ASEME methodology to build an AmI system based on agent technology; it also presented a novel software architecture for integrating several related technologies. ASEME has already been used in other real-world systems, such as modeling the behavior of robotic teams, an automated product pricing system, and a hybrid wind turbine monitoring system (www.amcl.tuc.gr/aseme/Publications.html).

There are specific advantages of the HERA approach compared with

previous works.[9,10] For example, user autonomy is increased and ambient assistance is automated.[7] Moreover, our system validation results show that the use of agent technology provides added value to personal assistance in AmI environments.

Our future work is related to integrating more sensors for a continuous flow of data when monitoring user behavior—the goal is to detect his or her state passively and inform family members or call for help during abnormal situations (such as when the user has fallen down and can't get up). This research calls for new types of agents that can handle wrapped sensors and event recognition at runtime. Fortunately, our original agent-based system modeling allows for the seamless integration of new agents and functionalities. ■

References

1. J. Nehmer et al., "Living Assistance Systems: An Ambient Intelligence Approach," *Proc. 28th Int'l Conf. Software Eng.*, 2006, pp. 43–50.
2. M. Wooldridge, *An Introduction to MultiAgent Systems*, 2nd ed., John Wiley & Sons, 2009.
3. P. Bresciani et al., "Tropos: An Agent-Oriented Software Development Methodology," *Autonomous Agents and Multi-Agent Systems*, vol. 8, no. 3, 2004, pp. 203–236.
4. F. Zambonelli, N.R. Jennings, and M. Wooldridge, "Developing Multiagent Systems: The Gaia Methodology," *ACM Trans. Software Eng. Methodology*, vol. 12, no. 3, 2003, pp. 317–370.
5. D. Harel and A. Naamad, "The STATEMATE Semantics of Statecharts," *ACM Trans. Software Eng. Methodology*, vol. 5, no. 4, 1996, p. 293.
6. N. Spanoudakis and P. Moraitis, "Modular JADE Agents Design and Implementation Using ASEME," *Proc. IEEE/WIC/ACM Int'l Conf. Intelligent Agent Tech.*, 2010, pp. 221–228.
7. J. Marcais, N. Spanoudakis, and P. Moraitis, "Using Argumentation for Ambient Assisted Living," *Artificial Intelligence Applications and Innovations*, vol. 364, 2011, pp. 410–419.
8. R. Colombo and A. Guerra, "MEDEPROS: The Evaluation Method for Software Product," *Proc. 15th Int'l Conf. Software & Systems Eng. & Applications* (ICSSEA), 2002; www.cti.gov.br/images/stories/cti/publicacoes/pdf/2002/evaluation_software.pdf.
9. J. Bravo et al., "Enabling NFC Technology for Supporting Chronic Diseases: A Proposal for Alzheimer Caregivers," *Ambient Intelligence*, vol. 5355, E. Aarts et al., eds., Springer, 2008, pp. 109–125.
10. Ó. García et al., "ALZ-MAS 2.0; A Distributed Approach for Alzheimer Health Care," *Proc. 3rd Symp. Ubiquitous Computing and Ambient Intelligence*, vol. 51, J. M. Corchado, D. I. Tapia, and J. Bravo, eds., Springer, 2009, pp. 76–85.

THE AUTHORS

Nikolaos Spanoudakis is a teaching assistant in the School of Production Engineering and Management at the Technical University of Crete (TUC), Greece. His research interests concern intelligent agents and multi-agent systems, with special focus on agent-oriented software engineering. Spanoudakis has a PhD in computer science from the Paris Descartes University (France). He's a senior member of IEEE and a member of ACM, the Hellenic Artificial Intelligence Society (HAIS), and the Technical Chamber of Greece (TCG). Contact him at nikos@amcl.tuc.gr; http://users.isc.tuc.gr/~nispanoudakis.

Pavlos Moraitis is a professor of computer science in the Department of Mathematics and Computer Science at Paris Descartes University (France), where he's also the director of the Laboratory of Informatics Paris Descartes (LIPADE) and head of the Distributed Artificial Intelligence Group (DAI). His research activity concerns both theoretical and applied research in the field of artificial intelligence, and specifically in the area of intelligent agents and multi-agent systems. Moraitis has a PhD in artificial intelligence from the Paris Dauphine University (France). Contact him at pavlos@mi.parisdescartes.fr; www.math-info.univ-paris5.fr/~moraitis.