In IAAI 04, San Jose CA, July 25-29, 2004, pages 852-857.

The Independent LifeStyle Assistant™ (I.L.S.A.): AI Lessons Learned

Karen Zita Haigh, Liana M. Kiff,
Janet Myers, Valerie Guralnik, Christopher W. Geib, John Phelps, Tom Wagner
Honeywell Laboratories, 3660 Technology Drive, Minneapolis, MN 55418
{karen.haigh,liana.kiff}@honeywell.com

Abstract

The Independent LifeStyle Assistant™ (I.L.S.A.) is an agent-based monitoring and support system to help elderly people to live longer in their homes by reducing caregiver burden. I.L.S.A. is a multiagent system that incorporates a unified sensing model, situation assessments, response planning, real-time responses and machine learning. This paper describes the some of the lessons we learned during the development and six-month field study.

1 Introduction

Historically, 43% of Americans over the age of 65 will enter a nursing home for at least one year. We have been developing an alternative: an automated monitoring and caregiving system called *Independent LifeStyle Assistant™* (I.L.S.A.) [9; 10; 12]. Researchers and manufacturers are developing a host of home automation devices that will be available in the near future. I.L.S.A.'s concept is to integrate these individual devices, and augment them with reasoning capabilities to create an intelligent, coherent, useful assistant that helps people enjoy a prolonged, independent lifestyle.

From January to July 2003, we field tested I.L.S.A. in the homes of eleven elderly adults. The I.L.S.A. field test was designed to complete an end-to-end proof-of-concept. It included continuous data collection and transmission via security sensors installed in the home, data analysis, information synthesis, and information delivery to I.L.S.A. clients and their caregivers. The test concentrated on monitoring two of the most significant Activities of Daily Life: medication and mobility. All ADL-based monitoring was performed by family caregivers.

This paper describes the system we built, outlines the field study, and then describes the major lessons we learned relating to AI technology:

- Agents
- Developing and making use of an ontology
- Automated Reasoning
- Integration

Haigh *et al* [9] describe many additional lessons learned, including client selection, system configuration, and usability.

2 System Description

The main goal of the field test was to demonstrate the complete cycle of I.L.S.A. interactions: from sensors to data transmission to reasoning to alerts and home control.

We selected our initial feature set based on their importance ranking, the ability to exercise the full range of technical capabilities of the I.L.S.A. architecture, and the need to learn more about a particular area. The ability to implement and appropriately support a robust test application was the final determining factor. The system we field tested had the following significant features:

- Passive Monitoring: basic mobility, occupancy, medication compliance, sleeping patterns.
- Cognitive Support: reminders, date/time of day.
- Alerts and Notifications: auto contacting caregivers (by telephone).
- Reports: summary reports of client behavior.
- Remote access to information (Internet or telephone)
- Control: modes (on/off).

Other capabilities and features were tested in the lab.

2.1 Architecture

Because clients age, and technology changes, I.L.S.A. had to be rapidly deployable, easy to configure, and easy to update. To meet these requirements, we decided to use an agent-oriented approach [10]. An agent-based architecture would provide modularity, distribution, functional decoupling, and dynamic discovery of capability. It would also make our ontology explicit. We selected JADE as our environment [1].

The agents in the system included device controllers, domain agents, response planners, and system management. One of each of the following agents were created for each human client: Medication, Mobility, Modes (on/off), Reminders, ResponseCoordinator, MachineLearning. The sketch in Figure 1 shows sample capabilities.

Exactly one each of the following agents was created on the server (one server for all the human clients): PhoneAgent, Platform, Database. In the research system, we explored task tracking (Section 5.1), machine learning techniques (Section 5.3) and several more domain agents.

The hardware employed in the I.L.S.A. field test consisted of readily available Honeywell home automation and control products. The Honeywell Home Controller served as the backbone for communicating sensor events out of the home.

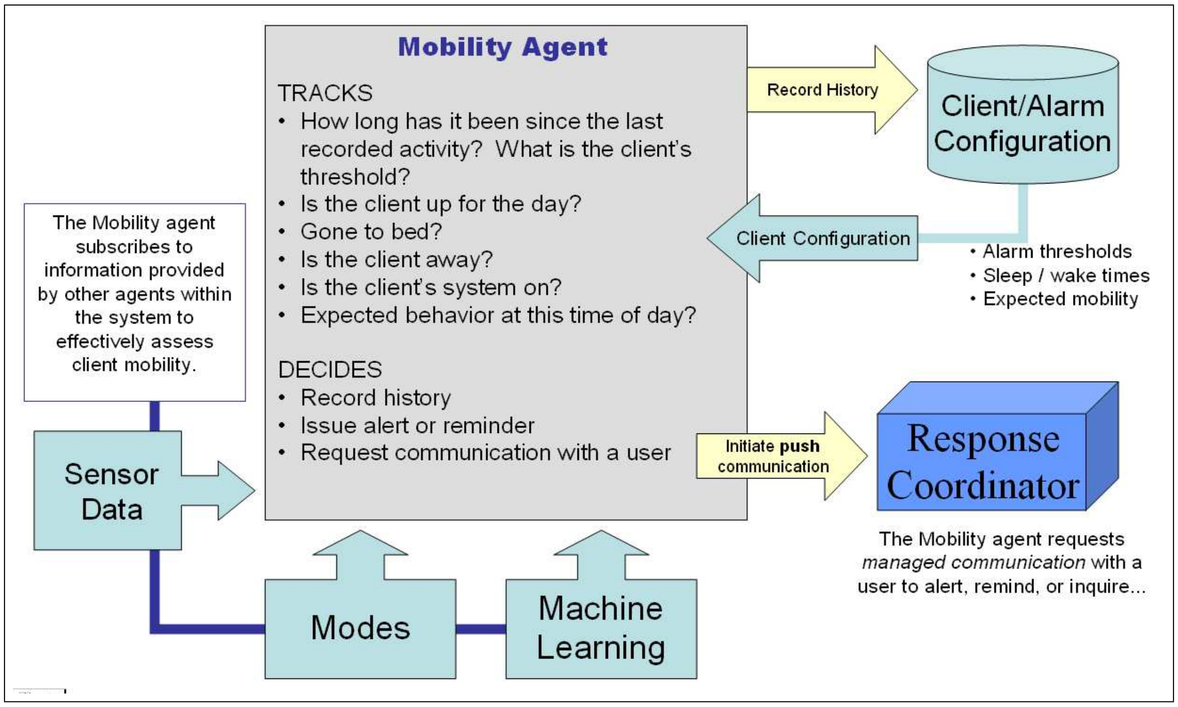

Mobility Agent
TRACKS
• How long has it been since the last recorded activity? What is the client's threshold?
• Is the client up for the day?
• Gone to bed?
• Is the client away?
• Is the client's system on?
• Expected behavior at this time of day?
DECIDES
• Record history
• Issue alert or reminder
• Request communication with a user
Record History
Client/Alarm Configuration
Client Configuration
• Alarm thresholds
• Sleep / wake times
• Expected mobility
The Mobility agent subscribes to information provided by other agents within the system to effectively assess client mobility.
Sensor Data
Initiate push communication
Response Coordinator
The Mobility agent requests managed communication with a user to alert, remind, or inquire...
Modes
Machine Learning

Figure 1: The Mobility agent tracks the client's activity.

2.2 Field Study Environments

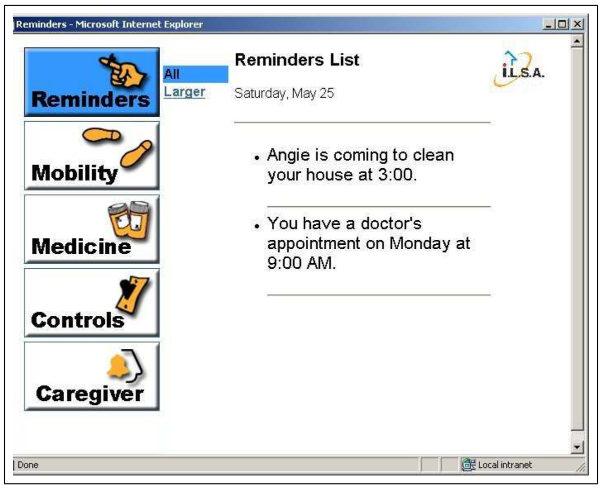
We installed I.L.S.A. in the homes of four system engineers, and eleven elderly clients. From July 2001 through December 2001 we installed four systems in engineers' homes, and focused on hardware configuration to determine which sensors were most effective. No reasoning components or user interfaces were included in these deployments.

Beginning January 2003, we installed I.L.S.A. into the homes of eleven elders and collected data through July 2003. We limited the number of sensors in the elders' homes for reasons of cost and concerns about privacy—for example, it would have been difficult to find appropriate test subjects who would accept a system with a toilet flush sensor. Each test home had from four to seven sensors, including one medication caddy and several motion detectors. Two installations had a contact switch and pressure mat at the exit door.

2.3 User Interface

Elderly clients were equipped with Honeywell Web Pads™ with wireless access to the Internet over a broadband connection. Through the Web interface, the elders could display reminders, medication schedules & status, mobility summary, on/off controls and information about their caregivers. Figure 2 shows a sample web page for the elderly client. I.L.S.A. could also deliver reminders to the elder by phone.

Caregivers could access I.L.S.A. data about their client/family member with their normal ISP Web connection. The caregiver Web interface allowed the caregiver to view and acknowledge alerts, view general ADL status (including historical trends for medication and mobility, view and edit prescription and medication schedule, and set up



Reminders List

Saturday, May 25

- Angie is coming to clean your house at 3:00.
- You have a doctor's appointment on Monday at 9:00 AM.

Figure 2: A sample webpage from the elder user interface.

scheduled reminders and personalized activity alerts.

Alerts and reminders could be delivered by telephone. In addition, a dial-in telephone interface allowed caregivers to get abbreviated status reports and record and schedule reminders for the elder.

3 Lessons: Agents

In addition to meeting I.L.S.A. requirements, we expected the agent-based approach to provide the following benefits [10]:

Multi-Person Development. Development of independent agents could be assigned to independent developers.

Scalability. We expected that The distributed architecture would support a much more scalable system.

Robustness and Reliability. Distributed processing and control would mean that the system would not crash with a local single point of failure.

Testing and debugging. Independent agents could be independently tested.

Agent-based approaches to system development are still relatively new. Robust infrastructures for applying this technology to a real-world system are not commercially available; research prototypes are not yet fully reliable and are mostly unsupported. There was considerable risk in basing the I.L.S.A. field test on this cutting-edge infrastructure.

I.L.S.A. was one of the first agent-based systems seen by "real" people outside the lab. As a result, we identified several risks and pitfalls in developing agent-based systems that had not been previously identified. We also describe new ways to address and mitigate these risks.

3.1 Lesson #1: Multi-Person Development

Since each I.L.S.A. agent performs a single role, we expected to facilitate rapid deployment of the system by assigning development of each agent to a separate software engineer. During the design stage we discovered that the agents could not be developed independently of each other. Even though each agent is responsible for a single task, many tasks are related to those of other agents. For example, when a *Medication* domain agent generates a reminder to take a medication, it is processed by the *ResponseCoordinator*, which coordinates interactions over multiple domain agents. It is then delivered to the client by the *Phone* agent. All three agents share a single goal—delivering a medication reminder to elder—and must be able communicate with each other *as well as the expected portions of the system*.

To support inter-agent communication, the development team needed to resolve such issues as communication protocols, recovery from failures or exceptions in agent conversations, and ontology development for semantic information exchange. Each time a protocol changes in one agent, many other agents also need to change.

In I.L.S.A.s tightly coupled system, the promise of independent development did not materialize. Agents needed to communicate with each other much more frequently than we had anticipated, and so resolving these issues required a great deal of coordination among team members and added considerable overhead to development time.

3.2 Lesson #2: Scalability

The I.L.S.A. system needs to handle a large number of clients, but current agent technology does not address scalability in any meaningful way. We noted three particular challenges for future agent system development.

First, to provide controlled access to client data, we implemented a database agent that performed all data read and write operations. While this approach insured database consistency, it also meant that the database agent could become a localized bottleneck. One approach to solving this problem is to have multiple, "specialized" databases.

Second, we had to determine how to properly scope each agent—hoping to build lightweight agents while keeping agent coordination and communication tractable. We therefore decomposed the message delivery task between the *ResponseCoordinator* and the device agents (see [15] for more). This innovative structure allowed us to decouple protocols for delivering different types of messages from the details of the message mediums. For example, the system could have multiple *Phone* agents and an *Email* agent.

Because reminders for different items (for example, two medications) can be issued at the same time, we designed the delivery protocols to multiplex such reminders for delivery as one message. We put this capability in the *ResponseCoordinator* agent so the system could handle multiple devices, each with its own dedicated agent. Because a reminder could be received by *ResponseCoordinator* right after it dispatched the previous reminder to the *Phone* agent, multiple messages were still delivered. To overcome this issue, we replicated the delivery protocol in the *Phone* agent, defeating the purpose of decoupling protocols. A *cancel* message would not have been an appropriate solution, because then it is possible that no message would ever be delivered.

Finding an effective scoping and decoupling of capabilities is extremely challenging, and becomes more so as the system grows. Our hope for "straightforward" scalability will not be possible until agent technology develops much stronger and more scalable mechanisms for enforcing (logical) protocols.

Finally, when we chose an agent-oriented approach, we expected to use a small set of agents for the basic functionality of each installation of I.L.S.A. Each installation could then be customized by adding specialized agents. As client requirements or technology capabilities evolved, agents could be replaced with different versions.

While all agent-based systems encounter complications with new agents, they are more severe in a tightly coupled agent system like I.L.S.A. We discovered that developing new agents required *extensive* re-development of the existing system. As new agents are added, new functionalities and interfaces need to be added to existing agents: you cannot simply plug in a new agent. These issues are compounded by multi-person development and also increase testing effort.

3.3 Lesson #3: Robustness and Reliability

One benefit of the multi-agent approach is that it is distributed; however, this approach does not preclude the system from having a single point of failure. Notably, the system has several agents to provide services for other agents. Some of these service agents (e.g. *Database*, the *ResponseCoordinator*, or *Platform*) are more critical than others; if one of them fails, the whole system fails. Failures of less critical service agents (e.g. *Phone*, *UnexpectedActivity*) severely limit functionality. If a "non-service" domain agent (e.g. *Medication*, *Mobility*) fails, other parts of the system still work, although this reduces the usefulness of the system. Redundant capabilities (both software and hardware) is one way to address this reliability problem; however, the designer will need to carefully consider how to not generate redundant (read “irritating”) messages to the users.

In I.L.S.A., the promise of inherent robustness from distributed computation did not hold true. Certain capabilities need to be centralized (e.g. communication with the elder) and this centralization is guaranteed to make system design difficult. An agent-based approach is unlikely to be appropriate when capabilities need to be centralized.

In a system where reliability is critical, it is important to design for persistence over restarts. For example, many of I.L.S.A.s domain agents needed a concept of recent history to make interaction decisions. If the system failed or was rebooted for some reason, agents had to reconstruct their history. Different agents reasoned over different windows of activity, hence only localized approaches to solving this problem are appropriate.

3.4 Lesson #2: Testing and Debugging

The requirement that every system needs to be thoroughly tested was noticeably more pronounced in the agent-based system than it would have been in a monolithic system. The ability to communicate “freely” with other agents meant that every possible interaction needed to be tested and verified; in a monolithic system, however, interactions between components are much more controlled, and testing can be focussed on the single point of change. With each new agent, the entire system had to be thoroughly tested to check behavioral and data coherence.

The extent to which the agents communicate and work together resulted in significant complications for debugging. Current agent technology does not provide adequate support to localize bugs: an error generated by an agent does not necessarily identify the root cause of the problem. Errors can propagate from agent to agent through communication channels, making it difficult to identify the agent at fault.

As a side effect of inadequate support for enforcing logical protocols throughout the system, we often found that we had to fix the same bug in multiple agents. It was also hard, with the multi-person development team, to know who had found time to implement the new protocol, and who had not. Often, this led to finding the same error during multiple, independent testing sessions, and therefore multiple, independent hunts for the same bug.

3.5 A Final Note

The agent-based approach promised a highly open and flexible system. We learned that this approach still requires a very rigorous software development process and many challenges are yet to be addressed by the agent research community. Agent based systems will need to focus much more strongly on scalability, debugging, and reliability issues before they will emerge from the lab.

We believe that the development issues we faced will be shared by any system in which agents work cooperatively toward a common goal. The more centralized that capabilities need to be, the more likely that an agent-based approach is inappropriate.

4 Lessons: Ontology

The Consolidated Home Ontology in Protégé (CHOP) serves two primary purposes. First, it is a common vocabulary for I.L.S.A.-related concepts, and their relationships. Second, in conjunction with a program code generator, CHOP produces an agent communication interface between I.L.S.A.’s agent-based system components.

CHOP is an ontology containing over 800 distinct concepts. It was developed with Protégé [7], a popular visual ontology construction tool. CHOP was derived from two upper ontologies, Cyc [3] and the Suggested Upper Merged Ontology (SUMO) [13; 14]. CHOP contains concepts including support for agent configuration, logging monitoring results to a long-term store, client and environment states, and status communication.

In addition to clarifying the meaning of terms and objects in a system, the power of a formal ontology is that it may be used as the basis to automatically generate portions of code that are otherwise tedious. I.L.S.A.’s inter-agent communication depended upon Java classes that were auto-generated from the ontology. Taken to the furthest extreme, many system artifacts can be automatically generated using a formal ontology, including communications interfaces in multiple implementation languages, database schema, and other formal interfaces such as database access routines.

Useful lessons we derived from the creation and use of CHOP include:

- Designing one ontology for multiple purposes may mean trading lack of duplication for a steeper development learning curve.
- Don’t waste ontological development effort supporting concept taxonomy or attributes that are not dictated by the application, even if they are relevant to the domain.
- In developing a taxonomy, be conscientious about cross-cultural compatibility.

While the use of ontologies is not novel in software design, existing ontologies did not cover the eldercare domain. While it is incomplete, CHOP represents a promising starting point for developing a reusable ontology for knowledge-based home assistance systems. We will be presenting CHOP to the Center for Aging Services and Technologies (CAST) [2] Electronic Health and Wellness Records task group. The CAST group is working on recommendations for the Health Level Seven (HL7) [11] medical record standard regarding long-term assisted care.

Researchers interested in obtaining a copy of the ontology can send email to the authors of this paper.

5 Lessons: Automated Reasoning

At the outset of the program, a major focus was on using high-level reasoning to provide an intelligent monitoring system. We intended to build three main components: situation assessment and task tracking, response generation, and machine learning. Experiments in the engineers’ homes validated our hypothesis that this domain is very complex and lends itself very well to advanced reasoning techniques.

5.1 Lesson #1: Situation Assessment and Task Tracking

A significant risk for a monitoring system like I.L.S.A. is accurate recognition of activity. Sensors are noisy, inaccurate, and low quality. It is a significant challenge to assess the situation and recognize what activities are occurring.

Our task tracking system was based on the Probabilistic Hostile Agent Task Tracker (PHATT) [5]. At the beginning of the program, we recognized that PHATT had a number of capabilities not available in other task tracking systems, making it uniquely suited to this domain. Continuing our research on I.L.S.A., we reached the following list of requirements for task tracking systems in this kind of domain:

Abandoning Plans: All people abandon plans at one time or another: they forget what they are doing, get distracted, or decide explicitly to abandon a goal.

Unobservable actions: Not all elders are willing to have their actions observed by an assistant system and may try to hide their actions. This unobservable action stream is a significant challenge.

Failed Actions: People will often try to achieve a goal but fail. This information is extremely useful in this domain.

Partially Ordered Plans: People frequently work on multiple goals at the same time. The task tracker must be able to recognize interleaved plans.

Actions used for multiple effects: Often one action can achieve multiple effects; the task tracker must be able to handle this kind of *overloaded* action.

World state: Different factors in the environment can significantly affect the likelihood of the elder adopting different goals.

Multiple hypotheses: One set of actions might be contributing to more than one plan. The task tracker needs to provide a ranked list of different possibilities.

In the lab, using data from the four richly-sensored engineer homes, we demonstrated the importance of each of these requirements. Geib [4] describes these requirements in more detail. Geib and Goldman [6] describe the specific extensions made under I.L.S.A. funding to PHATT to incorporate reasoning about abandoned goals.

The PHATT-based intent recognition component was removed from the I.L.S.A. field study because, in an effort to reduce the deployment cost for the elders in the field study, I.L.S.A. focused on a reduced set of low-cost sensors. To fully realize the potential of the task tracking system, more extensive and higher level sensor information is required.

5.2 Lesson #2: Response Generation

Response generation deals with 1) deciding *how* to respond to a given situation that exists with the elderly, and 2) *coordinating* all responses that may occur at any given moment.

Domain agents decide how to respond to a situation in a *context-free* manner. That is, they generate appropriate responses only within the context of their expertise; they do not take into account responses that other domain agents may be generating. Responses fall into four categories: reminders, notifications, alerts, and alarms. Each response category is defined by specific protocols and priorities that I.L.S.A. must follow.

Excessive and inaccurate alerts are a significant risk for this kind of monitoring system. In our field study excessive no-motion alerts were a major cause of client and caregiver dissatisfaction. In a few cases early in our test, they created a small amount of panic when an elder forgot to “turn I.L.S.A. off” when leaving the apartment. A secondary and very real risk is that the bad alerts will cause real alarms to be ignored (“never cry wolf”).

The context-free messages must be prioritized, timely, and coordinated. Messages cannot be lost in a flood, and elders (and their caregivers) must not be overwhelmed. We chose to coordinate responses centrally rather than giving domain agents distributed access to devices. The centralized approach allows us to ensure that submission of important messages is *context-aware*:

- All responses from all domain agents must be presented on one set of devices, to one set of recipients. If we allowed domain agents to access devices directly, the recipients could be overwhelmed. Imagine receiving a dozen phone calls for different reminders at 9:00am; we wanted to merge these reminders into a single call.
- I.L.S.A. must recognize message priorities. Allowing domain agents to have distributed access to devices could cause important messages to be lost or delivered too late. For example, if the elder were to fall, the emergency call to a caregiver might be delayed behind a reminder to eat lunch.

For more detailed discussion refer to Wagner *et al.* [15].

5.3 Lesson #3: Machine Learning

Success in this domain requires that I.L.S.A. capture complex interactions among devices, the environment, and humans, and be particularly responsive to constant changes. Currently these systems are very inflexible, and their initial configuration is labor intensive. To adapt to changes in the client or in the environment (inevitable in this domain), we must reprogram the system. Reprogramming adds to the cost of the system and presents an inconvenience to the client.

We explored three machine learning techniques to assess their impact on I.L.S.A.’s actual operating environment:

Patterned behaviour profiles [8]: build models of which sensor firings correspond to which activities, in what order, and at what time.

Unexpected Activity : raise alerts if activity occurs when it is probabilistically unlikely.

Schedules : learn schedule information for regular activities in the clients home (e.g. medication, wake/sleep, occupancy)

Results of our analysis showed that Machine Learning is a useful enhancement to even the simplest system. At a minimum, machine learning techniques can learn schedules and provide long-term activity trends. Given a rich sensor suite, machine learning techniques can learn complex models of the environment, the elder, other people, and even the effectiveness of its own devices. These models can then be used by the system to improve its assessments and responsiveness. Machine learning techniques allow a system:

1. to tune itself to the operating environment, greatly reducing the amount of tuning and knowledge acquisition required at setup.

2. to respond to changes in the users and the domain, directly reducing maintenance costs.
3. to capture the user's preferences, enhancing usability.

Two main barriers remain: evaluation and automatic incorporation of learned models into the system.

It was not possible to gather ground-truth information for I.L.S.A. because users were not willing to either note every activity or be constantly videotaped. Our evaluations were based on "eyeballing" results for plausibility and then observing whether changed system behavior was more or less acceptable to the elders and their caregivers. While short-term ground truth can be gathered, researchers will need to develop good techniques for long-term evaluation.

Currently, the elder-care industry will not accept a system that does not have a human-in-the-loop; many other industries have regulations that explicitly prohibit automatic modifications. Our approach of presenting the results to a human before incorporating them into the system was well-accepted but is unlikely to scale to a larger community of users in its current form.

6 Conclusion

I.L.S.A., in its form as an agent-based system, has been retired. Using an agent-based system, contrary to our expectations, *significantly* added to the development effort. I.L.S.A. had many capabilities that needed to be centralized, and therefore it is clear to us that pursuing a simpler route would have saved us time, money, and frustration—a single-threaded, component-oriented architecture may have been a better approach.

If we continue research on the Independent LifeStyle Assistant, we will create a new software platform using the knowledge we gained during this program. The largest technical barriers we perceive are:

- further development, testing and verification of intelligent automation within this domain;
- development of a more effective medication management system, including 360-degree pharmaceutical services;
- development of effective and comfortable methods of communication between the system and the elderly clients.

We view the I.L.S.A. program as a success because of the following significant achievements:

- We successfully prototyped a passive monitoring system for elders in their own homes.
- We have a much better understanding of what constitutes an *acceptable* monitoring system for elders and, in particular, disproved some of the assumptions made about "technophobic" elders [9]. Through our Knowledge Acquisition effort, we learned what factors affect elders' independence and identified technology opportunities. We improved the understanding of factors that impede the delivery and acceptance of assistive technologies, and also improved our ability to overcome these factors.
- We validated the importance of artificial intelligence technologies to support a broad customer base in widely varied and unstructured environments. Notably, we have validated the importance of machine learning as a technique to mitigate expensive installations and ongoing adaptation (Section 5.3).

Acknowledgments

The authors would like to thank the contributions of the entire Honeywell I.L.S.A. team, as well as Dr. Kathleen Krichbaum, United Health Care, Presbyterian Homes and Services, and LifeLine Systems, Inc. We also acknowledge the support of the Advanced Technology Program at the National Institute of Science and Technology, U.S. Department of Commerce under agreement #70NANBOH3020.

References

[1] F. Bellifemine, A. Poggi, and G. Rimassa. JADE: A FIPA-compliant agent framework. In *Proceedings of The Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM)*, pages 97–108, 1999. http://sharon.cselt.it/projects/jade/.

[2] Center for Aging Services Technologies. Available from: www.agingtech.org, 2003 [cited 3-Sept-2003].

[3] Cyc. Available from: www.cyc.com/, 2003 [cited 3-Sept-2003].

[4] C. W. Geib. Problems with intent recognition for elder care. In *Proceedings of the AAAI-02 Workshop "Automation as Caregiver"*, pages 13–17, 2002.

[5] C. W. Geib and R. P. Goldman. Probabilistic plan recognition for hostile agents. In *Proceedings of the FLAIRS 2001 Conference*, 2001.

[6] C. W. Geib and R. P. Goldman. Recognizing plan/goal abandonment. In *Proceedings of IJCAI 2003*, 2003.

[7] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of Protégé: An environment for knowledge-based systems development. Technical Report SMI-2002-0943, Stanford University.

[8] V. Guralnik and K. Z. Haigh. Learning models of human behaviour with sequential patterns. In *Proceedings of the AAAI-02 workshop "Automation as Caregiver"*, pages 24–30, 2002.

[9] K. Z. Haigh, L. M. Kiff, J. Myers, V. Guralnik, K. Krichbaum, J. Phelps, T. Plocher, and D. Toms. The Independent LifeStyle Assistant^{TM} (I.L.S.A.): Lessons learned. Technical Report ACS-P03-023, Honeywell Laboratories, 3660 Technology Drive, Minneapolis, MN 55418, December 2003.

[10] K. Z. Haigh, J. Phelps, and C. W. Geib. An open agent architecture for assisting elder independence. In *The First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, pages 578–586, 2002.

[11] Health Level 7. Available from: www.hl7.org/, 2003 [cited 3-Sept-2003].

[12] The Independent LifeStyle Assistant^{TM}. Available from: www.htc.honeywell.com/projects/ ilsa, 2003.

[13] I. Niles and A. Pease. Origins of the IEEE Standard Upper Ontology. In *Working Notes of the IJCAI-2001 Workshop on the IEEE Standard Upper Ontology*, August 2001.

[14] Suggested Upper Merged Ontology. Available from: ontology.teknowledge.com/, 2003 [cited 3-Sept-2003].

[15] T. A. Wagner, V. Guralnik, and J. Phelps. Achieving global coherence in multi-agent cargiver systems: Centralized versus distributed response coordination in I.L.S.A. In *Proceedings of the AAAI-02 Workshop on "Automation as Caregiver"*, pages 100–105, 2002.