System Concept for User Focus Management in Agent Worlds

W Pasman, 15 May 2003

Introduction

The usual way for a user to locate agents is to search a list of 'currently available agents' for the expected name or ontology [Uschold96]. Unfortunately this requires a lot of expertise of the user about which agent is capable for what, and knowledge about the ontologies that may be relevant for what he tries to do. When a lot of fine-grained agents are available, which we foresee in the near future, this approach will leave the user lost in a massive amount of agents.

The original Cactus proposal [Pasman02b] took KPN's Eileen system [YPCA02] and the original Mabel system [Kardol99] as a basis for supporting the user in his tasks. KPN's Eileen comprises a human secretary assisting the user in selecting and using services, and the Cactus proposal aimed at making an electronic version of the Eileen human secretary.

This approach was later dropped in favour of a stronger emphasis on distributed agent technology and ad-hoc peer-to-peer networking. The current concept thus is much more flexibile and capable of dynamically changing its behaviour, by inserting and removing agents from the user's active environment.

In our previous report [Pasman03b] we discussed possible support and management of user focus in huge agent worlds. This report briefly discusses a number of key issues. Then technical aspects of user focus management are discussed, and a first system concept or architecture to support user focus management is sketched.

Discussion of issues

There are several tightly related issues that put requirements on the required architecture. We discuss the expected use of multi-modality and distributed agent architecture.

Use of Natural Language

Natural language, being speech or written, seems to be especially useful when the user is not exactly sure about the type of service or agent he needs. Natural language offers many mechanisms to roughly specify what one needs, or what effect one is looking for. Once the user has found the appropriate agent and knows how it works, a fill-in form or menu interaction mechanism in many cases is more effective, accurate and time saving.

Speech recognition

Restriction of the speech recognizer vocabulary to the current task of the user is of prime importance to get an acceptable recognition rate.This suggests a per-agent specialized parser for maximal accuracy.

Unfortunately, there are several problems with a per-agent speech recogniser.

First, the speech recogniser probably has to be trained to the user's voice to improve recognition, and training on a per-agent basis seems not very acceptable. A single recogniser used by all agents, but set up with the vocabulary of the current agent, seems the way to go.

Second, a single agent may not have sufficient information to recognise a full user utterance. For example assume the museum is in a museum, looking at a mondriaan painting. If the user says "light

mondriaan painting" to get more light on the painting, the light agent will recognise the "light" but not "Mondriaan painting". Somehow agents have to share vocabulary as the task requires.
Third, in several cases it seems not possible to fix speech commands to single device. For instance, if the user asks "please light this painting", it is not clear whether the light agent or the painting agent has to handle his request. Instead, we think that a separate agent inbetween the two, something like a spotlight-agent or maybe even a light-a-painting agent, is needed. This is a critical notion, as it will dramatically increase the number of agents in the environment, and highly influence the system architecture.
Finally some "common English" has to be available without requiring every agent to implement a full English parser. For instance the user will usually say something like "please light my Mondriaan painting" instead of "light painting". The 'please' has to be converted into for instance a WANT speech act so that all agents can focus on their core business.

Multimodality
Multimodality, or the ability of agents to interface to the user via multiple modalities, seems hard to realize in an automatic way. In simple cases a simple converter may be good enough, but much depends on the interface requirements, complexity of the task, etc. For instance, in the EasyLiving intelligent house project up to seven mechanisms were available for even simple things such as switching on the light [Brumitt00]. Until further research makes clear how automatic conversion might be done, we think that explicit conversion has to be implemented for each agent separately. We think that with proper modularization, good tools and supporting agents such conversions can be realized with a few lines of code in most cases.

Heavy versus Lightweight Agents
A choice has to be made what level of granularity the agent system should have. On the one extreme, we could have a few very complex agents, for instance a personal agent capable of handling room lighting, heating, drinking coffee and coffee machines, payment, meetings, agendas, understanding both voice, gestures and direct manipulation (light switch presses or thermostate turns). On the other extreme we can have extremely simple agents, for instance a heating agent that only understand thermostate turns.
From the system architecture point of view and for ad-hoc insertion of functionality a highly modular design is required, which fits very bad with a monolithic heavy weight agent approach. To get modularity and flexibility, heavy weight agents have to be built on top of lighter-weight agents. The inter-agent communication will run over specialized protocols anyway, eg from the FIPA [FIPA01], and not via human-tailored interfaces. Therefore it seems reasonable to assume FIPA-restricted versions even of heavy agents, and a special user interface agent handling the user negotiation.
Additionaly, it seems reasonable to introduce several user interface agents on top of a FIPA agent, for instance one voice steered agent, one WIMP and one guesture interface.
Concluding, our architecture will contain both heavy and lightweight agents, as in Figure 1.

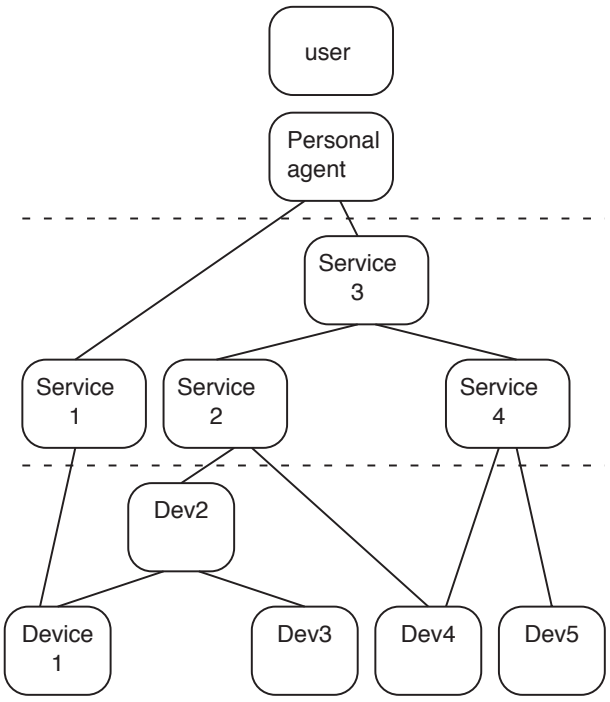

Figure 1. Close to the user a few 'fat' agents are available. These are using a larger number of service agents, which in turn can handle the device specific agents.

Resolving Referents

In comparison with traditional language processing techniques, the Cactus system has a much larger range of options to resolve referents. For instance when the user says "please light **this** painting", what 'this' points to can be determined not only from the previous utterances but from the full user focus as estimated. Table 1 shows a list of the factors involved., see also our previous report [Pasman03];

Table 1. A number of factors involved in user focus, which can also be used to resolve referents.

Focus Factor	Description
Mental focus	Which agent(s) were involved in last action(s)
Physical focus	What is the user's body doing (typing, driving)
Physical Location	Where is user (restaurant, at work, at home)
Focus of user eyes	What is user looking at
Voice location and direction	Is the user speaking, and in which direction
Focus history	Are there known patterns in the current focus behavior?
Salient environment properties and objects	Are there bright areas, highly visible objects, remarkable things that probably received user attention
Salient environment events	Are there flashing lights, loud sounds etc that probably have received user attention
Usual and current conversation partners	What are likely conversation partners?
Agenda	What's a probable user activity at this moment, according to his agenda
Recurrent tasks	What are recurrent and therefore probable user activities in the current situation.

Need for geometry and physics

Some have proposed to make the information available about the environment's geometry explicitly available [Brummitt00b]. This information can then be used to determine what the user is looking at, to locate nearby lights and displays. Even shadows and color changes due to reflections might be determined that way. Although not mentioned by Brummitt, such information could also be helpful by resolving geometric referents, like "the opposite wall" or "the back side".

However, resolving geometric referents in practice involves a lot more than just the geometric model and a bit of reasoning – the reasoning can become very complex very quick ([Soudzilovskaia01], [Aleven01])

Although resolving such referents will be useful in some situations, we think that this is not an essential part of our system. We think that the architecture we propose can be extended later to incorporate abilities to resolve geometric references. For instance, to answer queries like “What’s in this room” , the room agent could be able to answer this question as it has links to relevant devices in the room. Of course the scope of the answer will be limited to what the room agent programmer thought relevant for managing a room. But we already saw that for instance paintings may very well be included in the answer, as they are relevant for the lighting of the room. However pens may well be left out of the answer, as they may not be relevant for room management. However, the geometry based approach will probably miss pens as well, as the location of pens is not stationary and they are not important enough to track in real-time either.

Putting the geometry based approach one step further could involve adding rules of physics, such as physical requirements for actions. This way an entirely physics-based action planner might be constructed, for instance advising the user about technical possibilities to move a painting when he asks "can I move this painting". Pens might be tracked using camera-based tracking and knowledge of staying in place once inside a drawer. Although nice applications can be thought of, we will avoid this kind of system because we get the impression that it quickly amounts to making a detailed description of the 'real' world while the relevance for real user problems is quite limited.

Resource Management

There are a lot of issues involving resource management: quality of service, security, access rights, price of services, division of resources among agents, etc.

For managing quality of service, we earlier suggested to adopt the adaptive research contract system [Dijk00].Probably ARC has to be integrated with existing agent-agent negotiation schemes like contract-net [FIPA02]. However, the hard part of QoS negotiation is the abstraction and conversion of the parameters involved in the quality. We probably want to avoid the QoS issues altogether in Cactus.

Security concerns multiple aspects, such as safe transmission of data and proper handling of privacy-sensitive information. Safe transmission is merely a technical issue. But there is a lot of controvercy on 'proper handling' of privacy-sensitive information, and even on what is privacy-sensitive and what is not. Without such consensus it does not make much sense to put much attention on privacy issues.

Access rights involve the need of an agent to show some permit to a servicing agent, and the servicing agent will not act as requested until the permit has been shown. Sometimes the permit can be bought, at other times the permit is given by other agents. Sometimes the permit shows its validity by its mere existence, at other times the validity has to be checked with some authority. As with QoS issues, probably we should avoid access right issues.

Division of resources among agents is an interesting issue getting already a lot of attention in Cactus. For instance the WiFi discussion about how to motivate users to share their music, and the peer-to-peer discussion about how to get users share their radio channels to improve the network throughput. In our situation, interesting questions exist about who to allow to use the room speaker or wall display, and

how to share its use among multiple users. As an example,consider a big wall display used by multiple users. Each user could get a square part to put their own x-window screen in. There might be a little variation in the size of the squares, but essentially once all squares have been used no more users can use the display. The disadvantage is that extensive negotiation with the current screen users is needed to accomodate an additional user, and all their x-window settings have to be adjusted, windows to be moved, etc. Alternatively, the wall display could be a single x-window system, showing multiple windows for each user, according to his needs. If a user wants some window to be larger, he could just drag it larger (and he immediately starts paying more). This kind of sharing seems much more flexible to accomodate new users in a remaining hole on the screen, and also because resizing a window is much simpler than reconfiguring the screen it seems much more inviting to initiate a negotiation ("hey could you shrink your window a bit so that I can check my email?").
Concluding, I think there are more urgent topics to resolve before starting to work on resource management. There seem some nice and highly demonstrable possibilities in the resource-sharing area, but it is not clear to what extend these will help driving the research. There are clear advantages of enabling each agent to grab a piece of display as needed, and in the projected multi-user Cactus system I think many interesting possibilities would present itself if we would implement at least part of this.

Profile Management

Profile management is about managing known properties of the user, his preferences, and management of who has access to this information.
Some profile management is important, to get consistent system behavior and to avoid duplication of knowledge. Also it is important for the user to know where to look for his profiles. Without such management, user profiling has to be done on a as-needed basis, by the agent that needs the profile. Agents might have different ideas about the user's preferences. Knowledge will also be duplicated. This can cause inconsistency in the system behaviour. Also, it may be hard for the user to change his profile, as he will have to check several agents involved. It may even be the case that the agents he wants to check are not available at the moment he is trying to check them, for instance because he (or the agent) changed location.
Our proposed architecture did not impose an obligatory profiling system. However, we think that this will not lead to chaos. First, we expect that comparable agents will communicate relevant user information. Furthermore, higher-level agents (such as a display manager agent that picks the nearest display suiting the user's need and using it for interfacing) can distribute user preferences to the displays. The display settings even could be part of the requirements posed by the display manager, and displays not capable of meeting the setting requirements might be ignored by the manager. Finally, it is important to note that even a central profile manager can not guarantee consistent system behaviour as agents are always free to handle things themselves.
At a more detailed level, profile-related problems can appear as well. For example, assume the user wants to get to Amsterdam and was connected to the 9292 public transport information website. He just entered all travel details and got a trip plan. Now he asks the sytem “and how about going there by car”. It now would be nice if the system could tell the car route planner program about the relevant travel details that the user just gave to the public transport planner. In our system the logical approach to solve this problem is to have a kind of general-transport-planner agent that keeps track of the given info, and knows how to translate info to the public transport system and the car route planner.

System concept

The system we propose closely follows the FIPA recommendations [FIPA]. FIPA describes how agents communicate using an Agent Communication Language (ACL). ACL is based on speech-act like mechanisms. Agents can always refuse requests, although agents are always expected to reply. Communication runs along pre-defined protocols, defining the steps to be taken towards an agreement.
FIPA does not define which kinds of agents have to be made available in order to create an environment matching the user's needs. We will define which agents are needed, which knowledge has to be at an agent and what the semantics are of the resulting network.
Furthermore, FIPA does not define how the user is going to communicate with an agent. Some older FIPA documents proposed an intermediary agent doing a conversion between for instance speech and the ACL [FIPA]. But the latest definitions totally ignore the user interface side of the problem. Therefore, we here define how we will connect FIPA agents to the user.

Agents

The agents to be introduced in the system should match the speech acts that the user would need if he adresses the service he needs. Also there are separate agents for interfacing between the speech act and the actual user utterance (speech, texture, button, ..). Furthermore we assume an agent representing each relevant (physical or service) object. This is a cryptic definition, and is meant mainly to avoid introduction of agents without direct functionality.
Let's start with some examples. In the examples we will hit on a number of protocols and intermediaries, we hope the user gets a general idea here. Details will be worked out further below.
Assume that our system has to be capable to handle the utterance "light Mondriaan". Assuming that there is a spot light and a painting in the environment that would be involved in handling this user request, our definition says we have a spotlight agent and a painting agent. Furthermore, our definition says that we then need an agent for speech acts involving lighting of the mondriaan painting. So we also have a spotlight agent that could be named light-the-painting agent. Finally we have an voice-interface-for-light-the-painting agent, that can accept voice and convert it into the speech act which would be something like (REQUEST action:light object:painting24). Figure 2 shows a picture of the situation.

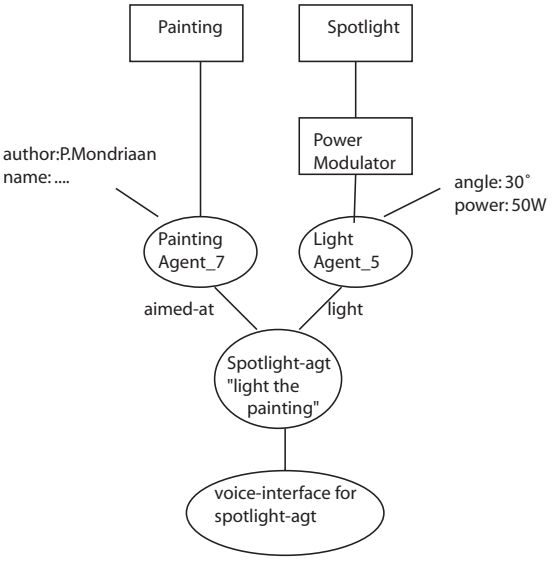

Figure 2. Resulting agents for handling "light Mondriaan".

As another example, consider "show map". Assuming a display in the user's environment, there is a display manager agent. Furthermore, there must be a map agent, which can get maps fitting the user's

current needs. It's the map agent that has to decide what map to show, but further below we will define some structure and supporting agents that can help making this decision. There also is a show-map-agent, that can get a map from a map-agent and convert it for display using a display manager agent. Finally there is an voice-interface-for-show-map-agent that converts this and similar user requests to a speech act for the show-map agent.

Interface agents
Interface agents are agents totally specialized in translation between one or more FIPA agents and the user. As we already saw in the previous section, there are voice-interfaces, but similarly there can be WIMP interfaces, keyboard interfaces and gesture interfaces. For the moment we propose an interface agent to be specialized in a single modality, but maybe multimodal agents show practical in the end.

Personal Agent
The personal agent functions as a 'default' agent to talk to and interact with. It can tell the user about problems, about environment properties (which agents are there?), can help adding or removing agents to the 'active agent' list. It keeps connected to microphones and displays in the user's neighbourhood. It queries the service matcher (below) as needed, and updates the user focus agent about the active agents.

User focus agent
Several agents can highly benefit from knowing the user focus as defined in the section 'resolving referents' in the discussion section above. We assume that there is a user focus agent keeping track of the focus aspects discussed there.

Active agents
There is an overall manager for the user, keeping track of the agents that the user approved for use. These approved agents are called the active agents. Maybe some mechanisms are needed so that the user can switch between sets of agents, for instance when he leaves his work he may switch from the 'at work agent set' to the 'private agent set' .

Service Matcher
In order to find an appropriate agent for a user, we assume that the user gives a command or question to a 'service matcher' agent. The service matcher will forward the user's utterance to the NL-interface agents (see below) that are currently active in the user's environment. The service matcher asks all these agents to assess the utterance, and the agents return an indication (some value between eg 0 and 1) whether they think the user is asking or adressing that particular agent.
The service matcher does not need to ask all NL-interface agents. Instead he can stop as soon as one with sufficient high confidence is found. Also he can consider the current user focus.
If there is a clear 'winner' (a single agent with very high confidence), the service matcher can subsequently ask that particular agent to handle the request. This fully enables the agent, and authorizes him to access I/O devices to get a more tight collaboration with the user.
If there are multiple highly confident agents, the service matcher may try to pick the best one based on user preferences, or alternatively it may ask the user to pick one.
If there are a few moderately confident agents, the service matcher probably has to ask the user for some more information. This is an interesting area for more research.
If there are no confident agents, the service matcher has to explain that no service can be found matching the request. Maybe he can give general hints on how to proceed.

NL-interface agents

Natural language (NL-)interface agents are interface agents specialized in understanding and converting natural language from the user. In our system their typical job is to assist the user in locating the appropriate agent for his job or task. NL-interface agents may be used both by speech- and plain text (keyboard, notepad) interfaces. Figure 3 extends Figure 2 with some NL-specific agents and data.

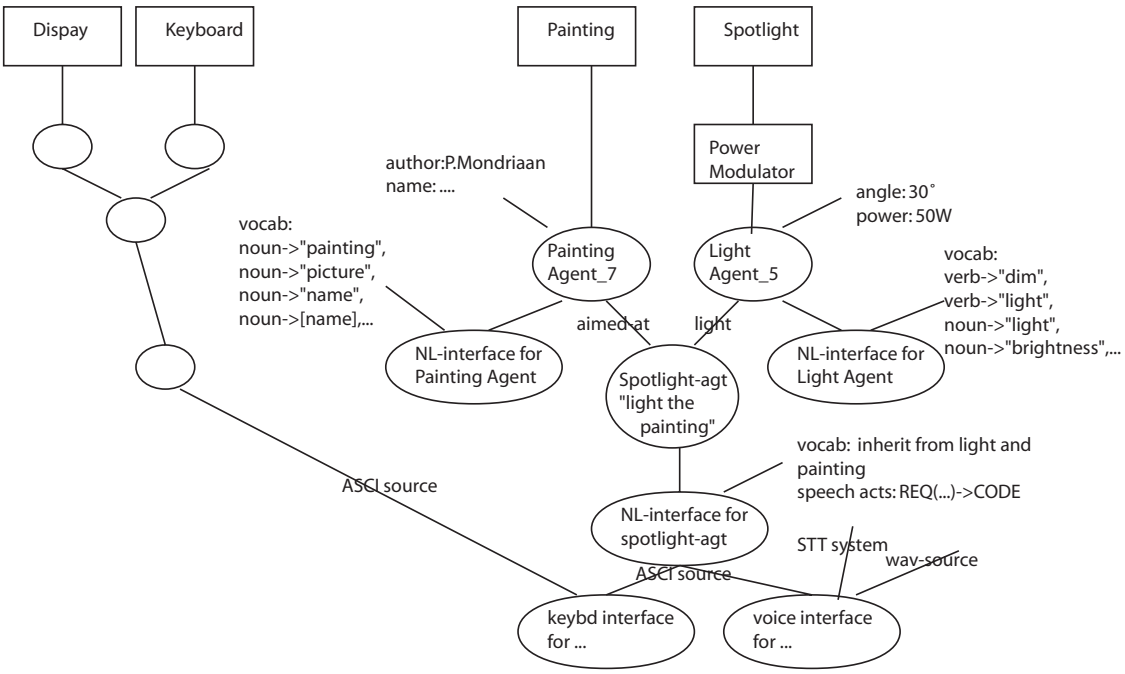

Figure 3. As Figure 1 but with more detail on the interfacing agents and voice structures.

For the NL-interface agents, we need extra functionality to support recognition of the user's utterance. Agents only recognize the words in their own expertise area, for instance the painting agent will recognize the name 'mondriaan' but not the word 'light'. The light-the-painting agent is a spotlight agent that knows about beaming light, but not about the painting. We propose that the spotlight agent collects the vocabulary from the 'aimed-at' object and from the light object. It can add its own words like 'aimed at' to the vocabulary, and the combined vocabulary is then used to attempt to recognize the user's utterance.

More specifically, each NL-interface agent imports a general language to speech act parser, and adds its own (presumably context-free grammar) rules to it. It would be nice if language can be loaded in steps, for instance if the user asks "I would like to book a trip to Antwerp", the travel agent might first try to parse with a general words like "book" and "trip", and only load the list of all places (including "Antwerp") after encountering "to" and some not-yet understood garbage.

The translation would then run in a number of steps. Consider "Please light this Mondriaan" being parsed by the NL-interface-for-spotlight agent. First, the type of speech act would be determined, giving something like (REQUEST "light this Mondriaan"). Language rules should clarify that light is meant as verb here, giving (REQUEST action:light "this Mondriaan"). "this" triggers a special procedure, checking neighbouring objects for the word "Mondriaan" (which needs to have some support for natural language, to be determined). The procedure returns something like painting_24, and the parse has become (REQUEST action:light object:painting24). Similar special procedures seem necessary for other referents like 'that', 'my', etc. The exact procedures in this translation need more research.

The voice-interface agent is converting user microphone input into plain text for the NL-interface agent. If possible, all voice-interface agents should use the same speech recogniser, so that the recogniser can properly learn the user's voice characteristics over a wide range of vocabulary.

Generic IO agents

The user may be moving around, introducing a need to easy switch between displays, microphones, speakers, keyboards etc. Instead of having each agent keeping track of nearby displays, keeping track of the various display protocols, properties, settings etc, we propose that each agent just communicates via a generic display, speaker etc agent.That generic agent keeps track of the best current IO device (nearby, in front of user, etc) and forwards all IO to it.

Discussion

This section discusses some problems and might-be problems.

Goal, Plans, Scripts

Translation of for instance “let’s get some hot coffee” would translate into some speech act like (REQUEST action:buy obj:(coffee mod:some mod: hot.)). Nearby coffee machines will respond, but the local coke machine should not. A drawback of such an approach is that we can not reason about user goals and plans. For instance the user may want coffee to get awake, and the system might know about maintenance work on the water pipes today and therefore no coffee may be available. In that case the system might propose to get some coke can. And even better the system might have warned the user at home already to pack some hot coffee. Those alternative solutions to fulfil the user plans are out of the scope of the proposed architecture.

An approach that seems to fit reasonably well in the proposed architecure is the notion of scripts. For instance a ‘restaurant agent’ might be able to recognise essential ‘restaurant script’ steps such as searching for a restaurant and being hungry. The agent might then become active or become responsive to user questions.

Scope of language understanding

Questions like “let’s clean the coffee I spilled on the ground” probably need a cleaning agent somewhere for answering that question. If that agent would be introduced into the system to answer the question, we suddenly also need detailed info on floor properties to be filled in by the user, such as “it’s a wooden floor”, “dark blue”, “tiles”, “synthetic”. Hopefully this info keeps available for the next time. But the cleaning agent may be ‘rented’ for instance from a cleaning expert site. For reuse in other situations, would be nice if the information is stored in a user profile database instead of at the rented agent’s site.

Other questions rise even more serious problems and imply a number of underlying uncertainties. For example, “can I remove this painting” implies a lot of questions like “do I have all tools needed to move it?”, “Is it allowed to move it?”, “Is it physically possible to move it?”, “would it be nice to move it?”, “would people starting complaining if it is moved?”, “who can give me permission for it?” and “please tell me how to arrange to move it”. It is not clear what kind of agent would handle such questions, and whether it would be a single agent or multiple agents.

It is not clear which speech acts an agent should be able to handle. For instance, should the spotlight-agent be able to answer the question“Which lights light this painting?”? More research and hands-on experience is needed to bring clarity on this issue.

Pro-activity

Pro-activity is the ability of an agent to take actions on its own. If the agent is using the user's resources for these actions (including the use of the user's processors, storage space, money, etc), the agent probably should have a good reason to do so, and the user should be able to ask the agent about its activities. More research is needed, amongst others to determine the types of pro-activity and the required resources, to find out which agents are pro-active and when, and how the user determines the amount of pro-activity (which is highly related to trust [Maes97]). There seems to be a tight relation between pro-activity and recognition of the user's goals and plans. As discussed in the previous section, the presented architecture proposes to use high-level agents recognising chains of user actions and events that might imply such goals and plans, and then take pro-active actions.

Conclusions

We proposed an architecture framework that can support the user in navigation and service discovery in a network of agents. Natural language is used as a powerful means for the user to describe roughly what he is looking for, as input for the system to locate the agent(s) that might be able to help the user. The framework describes which agents are needed in the environment and how the agents are queried for their usefulness regarding the user's problem. The framework is based on the FIPA proposals for intelligent agent networks, which should enable other multimodal interaction techniques besides natural language.

The proposed architecture will be able to understand only a very limited subset of natural language. Research is required for instance to find out which subset is required to realize sufficient user support, how the user can understand the limits of the system, and to determine how useful the framework is in practice.

References

[Aleven01] Aleven, V., Popescu, O., & Koedinger, K. R. (2001). A Tutorial Dialogue System with Knowledge-Based Understanding and Classification of Student Explanations. 2nd IJCAI Workshop on Knowledge and Reasoning in Practical Dialogue Systems (August 5, Seattle). Available Internet: http://www-2.cs.cmu.edu/~aleven/publications.html.

[Brumitt00] Brumitt, B. L., Meyers, B., Krumm, J., Kern, A., Shafer, S. (2000). EasyLiving: Technologies for Intelligent Environments. Handheld and Ubiquitous Computing, 2nd Intl. Symposium, September, 12-27. Available Internet: http://research.microsoft.com/barry/research/index.htm.

[Brummitt00b] Brumitt, B., Krumm, J., Meyers, B., & Shafer, S. (2000). Ubiquitous Computing & The Role of Geometry. IEEE Personal Communications, 7 (5. October, Special Issue on Smart Spaces and Environments), 41-43. Available Internet: http://research.microsoft.com/users/barry/research.

[Dijk00] Dijk, H., Langendoen, K., & Sips, H. (2000). ARC: A bottom-up approach to negotiated QoS. Proc. WMCSA'2000 (7-8 December, Monterey, CA), 128-137.

[FIPA] FIPA (2003). The Foundation for Intelligent Physical Agents. Available Internet: www.fipa.org.

[FIPA01] FIPA (2001). FIPA Communicative Act Library Specification. Foundation for Intelligent Physical Agents, Concord, CA. Available Internet: http://www.fipa.org/specs/fipa00037.

[FIPA02] FIPA (2002). FIPA Contract Net Interaction Protocol Specification. Available Internet: http://www.fipa.org/specs/fipa00029/.

[Kardol99] Kardol. J. J. (1999). Bring Mabel Back. Undisclosed Technical Report RA-99-31287, KPN Research, 1998.

[Maes97] Maes, P. (1997). CHI97 software agents tutorial. Tutorial, CHI'97 conference. Available Internet: http://pattie.www.media.mit.edu/people/pattie.

[Pasman02b] Pasman, W. (2002). Speed of FlyJacket Grabber. Internal Report, TTE Department , VTT Helsinki. Available internet: http://graphics.tudelft.nl/~wouter.

[Pasman03] Pasman, W., & Jansen, F. W. (2003). Comparing simplification and image-based techniques for 3D client-server rendering systems. IEEE Transactions on Visualization and Computer Graphics, 9 (2), 226-240.

[Pasman03b] Pasman, W. (2003). User Focus Management in Agent Worlds. Internal Report, April, Cactus Project, Delft University of Technology. Available Internet: http://graphics.tudelft.nl/~wouter/publications/publ.html

[Soudzilovskaia01] Soudzilovskaia, N. (2001). Visual Presentation and Interaction in a Multi-Modal User Interface. Project report for the post-graduate course AIDE, Faculty of Industrial Design, Delft University of Technology.

[Uschold96] Uschold, M., & Gruninger, M. (1996). Ontologies: Principles, Methods and Applications. The Knowledge Engineering Review, V.11, N.2, 1996. Available Internet: http://citeseer.nj.nec.com/uschold96building.html.

[YPCA02] YPCA (2002). Eileen: Your Personal Call Assistant. YPCA, Leidschendam, the Netherlands. Available Internet: http://www.ypca.nl.