ACTIVE, A PLATFORM FOR BUILDING INTELLIGENT OPERATING ROOMS

D. GUZZONI[1], C. BAUR[1], A. CHEYER[2]

[1]VRAI Group – EPFL – 1015 Lausanne – Switzerland
[2]AIC – SRI International – Menlo Park, CA – USA

Today computers are part of the standard equipment of modern surgery rooms. They assist surgeons in performing complex procedures that would not be possible otherwise. However, despite the availability of more powerful and complex computer systems, their user interfaces have not been adapted to fully leverage their potential. A new type of software, behaving as an independent intelligent assistant, is needed to better assist surgeons and their staff. Building an intelligent assistant is a difficult task that requires expertise in many fields ranging from artificial intelligence to core software and hardware engineering. We believe that providing a unified tool and methodology to create intelligent software will bring many benefits to this area of research. Our solution, the Active framework, introduces the original concept of Active Ontologies to model and implement intelligent applications. Based on suggestions and constant evaluations from surgeons, an Active based assistant for endoscopic neurosurgery is under development. Using natural modalities such as speech recognition and hand gestures, it enables surgeons to interact with computer based equipments of the operating room as if they were full active members of the team. In a broader context, Active aims to ease the development of intelligent software by making required technologies more accessible. It will help foster research innovation, easier development cycle and deployment of this new type of applications.

INTRODUCTION

Although computer systems have grown in power, access more networked content and services, computer interfaces have not changed. Conventional user interfaces with simple direct manipulation commands are no longer sufficient to fully leverage such rich and dynamic environment [1]. The medical field is no exception.

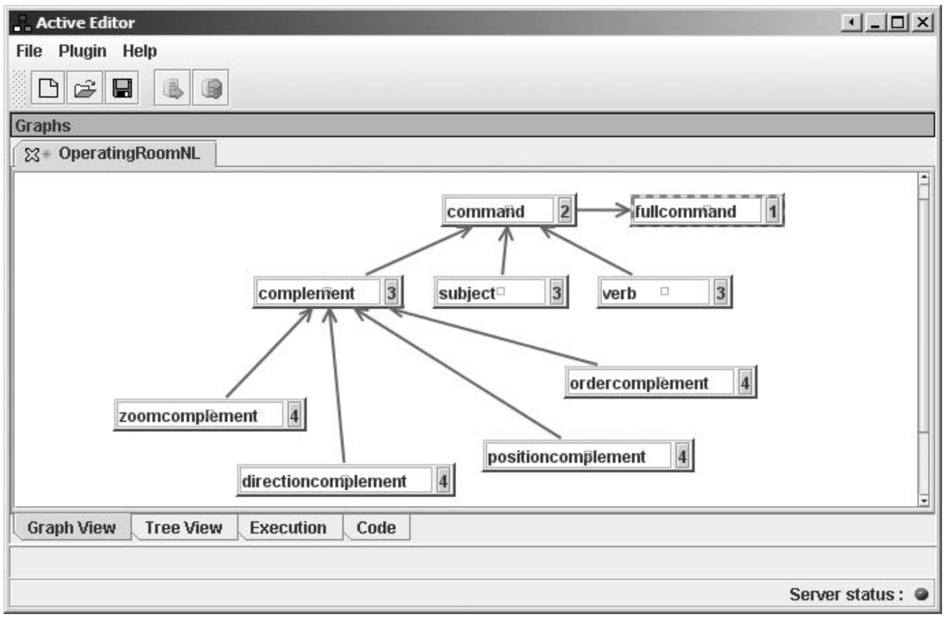

Figure 1 : Active Editor

Computers are now part of the standard equipment used in modern surgery rooms. To fully leverage this new context, modern software systems should behave as intelligent assistants able to observe and sense their environment, for instance human inputs, to analyze a situation by mapping input senses into a model of what tasks and events may be happening [2]. They would then understand and anticipate what the user might need to finally act to produce relevant and useful behaviour. The development of intelligent assistants requires expertise in many ¯fields [3].

Perception of human activities is typically based on techniques such as computer vision or speech recognition. Understanding the meaning of input signals, is performed by natural language processors, dialog systems or activity recognition mechanisms. Reaction, decision making strategies and complex task execution are the responsibility of planning systems. Finally, as planning unfolds various actions are taken by the system. Based on their nature and purpose, intelligent systems act through a wide range of modalities. They communicate with humans, gather information or physically change their environment. Designing and implementing intelligent assistants software is also a difficult task.

Due to the variety and complexity of technologies required, intelligent assistants are made of a collection of components written in many different programming languages. Connecting various heterogeneous programs, sometimes remotely, requires strong technical knowledge and careful deployment policies. Testing and debugging distributed heterogeneous

systems is also a complex task. To identify and correct bugs, events and associated values need to be tracked from one component to another. Finally, combining many different approaches, tools and technologies limits the overall performance and extensibility of the system.
We believe that providing a unified tool and methodology to create intelligent software will solve many of the problems described above and bring many benefits to this area of research. It will allow more researchers and engineers to work in the field by providing a bridge between core AI technologies and practical engineering.
This paper introduces our implementation of this vision, the Active framework. The next section is dedicated to related work on building intelligent assistants. The section *Active Framework* outlines the Active original concepts, architecture and current implementation. The next section presents how the Active framework is used to implement an intelligent assistant in the context of neurosurgery. Finally, a conclusion presents directions of our future work.

RELATED WORK

By definition, intelligent interactive systems are based on various AI techniques.
Relevant efforts related to our research can be classified into three categories. First, the area of interface agents aims at creating intelligent user interfaces to assist humans in specific domains [4]. For instance, the Internet is an environment where intelligent assistants can leverage a vast amount of information and services to help users with complex tasks [5]. Scheduling meetings, managing an agenda and communicating also represent applications where intelligent assistants are relevant [6].
Intelligent assistant are also relevant in the domain of heterogeneous smart spaces, instrumented rooms able to sense their environment and act upon events and conditions. In the surgical field, modern operating rooms are becoming such smart spaces. Many components can now be connected and controlled so that intelligent assistant software can be deployed to assist surgeons and their staff. Existing smart spaces projects are designed and optimized for specific domains, implemented using proprietary frameworks and methods. Our goal is to provide a more generic intelligent system toolkit, composed of a suite of tools and methodologies to rapidly design and deploy complex software into smart spaces.
Our work also relates to the field of multi agent framework research. In this area, heterogeneous existing AI based components are turned into agents able to form communities working together with humans to help them solve problems. In this context, the open agent architecture [7] OAA introduces the powerful concept of delegated computing. Requests and plans are delegated to a facilitator in charge of orchestrating actions based on declared capabilities of agents. Thanks to its ease of deployment and clean design, OAA is used in a large number of projects. Though very powerful, OAA does not provide a unified methodology to create intelligent systems. It rather provides a

framework where heterogeneous elements, written in many programming languages, are turned into OAA compatible agents to form intelligent communities. Similarly, the Retsina [8] framework is advanced multi agent architecture to build distributed intelligent systems. It is based on four classes of agents. Interface agents that interact with users, task agents that carry out plans, information retrieval agents and middle agents to help match agents that request services with agents that provide services. Though very efficient in producing independent reactive behavior, Restina would not be suited as a unified methodology to implement basic AI components such as natural language processors or multimodal fusion engines. In addition the design of Retsina uses different formalisms for communication, domain representation and reasoning technique. In contrast, our aim is to use the same formalism for all intelligent assistant aspects.

Finally, undertaking tasks on behalf of a user and attempting to understand what actions are being carried out involves planning. BDI based systems [9] provide goal oriented reactive planning in dynamic and partially known environments. Beliefs represent the model and state of the world and a plan library defines how to achieve goals. Intentions are activated plans elected and picked from the library to reach some goals. The list of intentions is constantly evaluated with beliefs, thus providing a reactive behavior to the system. Many BDI implementations [10] [11] are available and have proved their relevance in the field of intelligent systems. BDI based engines would be well suited to be the core of our research, where dynamic decisions need to be made to respond to an event. Their design is nevertheless constrained to dynamic planning and would not be suited to implement tasks such as natural language processing or modality fusion.

ACTIVE FRAMEWORK

1. Conceptual Overview

Our solution, the Active framework, provides a unified tool and methodology to eases the development of intelligent software. Active is based on the original concept of *Active Ontologies*, used to model and implement applications. A conventional ontology is defined as a formal representation for domain knowledge, with distinct classes, attributes, and relations among classes; it is a data structure. An Active Ontology is a processing formalism where distinct processing elements are arranged according to ontology notions; it is an execution environment. An Active Ontology is made up of interconnected processing elements called *Concepts*, graphically arranged to represent the domain objects, events, actions, and processes that make up an application. Concepts communicate with each other through channels, passing state information, hypotheses, and requests.

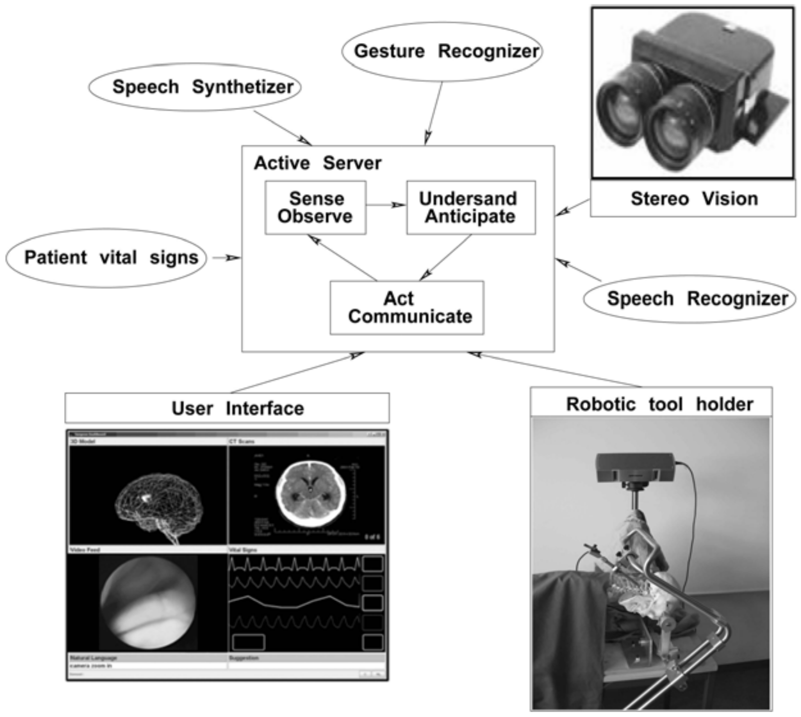

Figure 2 : Active Application Design

2. Technology

The Active framework implementation is a Java based software suite designed to be extensible and open. The Active Editor (Shown in figure 1) is a design environment used by developers to model, deploy and test Active applications. The Active Server is a scalable runtime engine that hosts and executes one or more Active applications.

A plug-in mechanism enables researchers to package AI functionality to allow developers to apply and combine the concepts quickly and easily. To ensure ease of integration and extensibility, components of the Active platform communicate through web service (SOAP) interfaces.

3. Active based application design

An Active powered application is composed of one or more Active Ontologies deployed and executed on the Active server and a community of sensors and actuators integrated as SOAP web services (See figure 2). Sensors (user interface, speech recognizer, stereo camera or any physical measuring probe) report events captured in the environment through the SOAP interface of the Active server.

In response to incoming events, an Active Ontology in charge of natural language interpretation attempts to construct structured commands. Such Active Ontology (See figure 1) defines the structure of valid commands and, within the same unified context, specifies processing rules to turn the static ontology-like domain definition into a dynamic execution environment.

An Active Ontology in charge of natural language interpretation is made out of two types of concepts: *sensor* concepts and *node* concepts.
Sensor concepts are specialized filters to sense and rate incoming events about their possible meaning. A rating defines the degree of confidence about the possible meaning of the corresponding sensed signal. Typically sensor concepts generate ratings by testing events ordering and if their values belong to a known vocabulary set. Sensors use channels to report their results to their parents, the *node* concepts.
There are two types of node concepts: *gathering* nodes and *selection* nodes. Gathering nodes create and rate a structured object made out of ratings coming from all their children. Selection nodes pick the single best rating coming from their children. Node concepts are also part of the hierarchy and report ratings to their own parent nodes. Through this bottom up execution, input signals are incrementally assembled up the domain tree to produce a structured command at the root node.
For instance, when the surgeon says: *"endoscope zoom in"*, the sequence of words *"endoscope"*, *"move"*, *"in"* will be submitted to the network. Each word is rated by the sensors of the network. *"endoscope"* will be rated as a *subject*, *"move"* as a *verb* and *"in"* as a *zoom complement*. The node complement is of type selection and picks the best rated value coming from its children. At the top of the network, the node command is of type gathering and assembles values from its children to create the final command.
Since sensors report events to the Active server through a web service interface, they can be heterogeneous, distributed and easily added. Active is a test-bed for multimodal applications where multiple sensors can contribute to make up a command.
For instance, a surgeon can say *"endoscope, follow my tool"* while gesturing to the left. The speech recognizer will contribute by reporting all recognized words and the gesture recognizer will report a gesture going from left to right. The language processing Active ontology, using its bottom up network of concepts, will assemble these fragments to generate a full command.
Concepts remember their current ratings, therefore the dialog context between the user and Active is maintained. After successfully issuing the command *"endoscope zoom in"*, to further control the zoom factor the user can simply say *"in"* or *"out"*.
Once a structured command has been generated at the language processing stage, it is passed to another Active Ontology in charge of validation and resolution. The incoming command will be deconstructed, following a top down scheme, to verify that each element is valid and semantically correct. Complete and valid commands are processed by a final stage, implemented as another Active Ontology, will perform actions and communicate.
Since Active applications interact with their environment through a set of loosely coupled services, actuators are not known at design time and have to be dynamically chosen at runtime based on their availability, the environment context and user preferences.
This concept of delegated computing [7] is implemented by another specialized

Active Ontology. Registered service providers are rated and picked at runtime by a delegation broker. As an example, if a message has to be communicated, the delegation Active Ontology will analyze the current situation to decide which service provider is best suited to do the job. Selection is based on many factors such as dialog context, user preferences, location, reliability or cost. Service integration through a delegation mechanism provides a powerful plug and play approach where components can be dynamically integrated.

NEUROSURGERY INTELLIGENT ENVIRONMENT

Following the methodology described in the previous section, an intelligent operating assistant for neurosurgery is under development. The system is implemented as a multimodal system allowing surgeons to retrieve and manipulate pre-operative data (a set of CT scans and a reconstructed 3D model of the area to operate). In addition, live images coming from a powered image source (endoscope or microscope) are displayed along with vital patient information. Surgeons and their staff interact with the system by a combination of hand gesture using a contact-less mouse [12] and voice recognition. Commands are issued to control the powered endoscope, navigate through pre-operative data and choose which information to show on the main display. The prototype is implemented over five Active Ontologies deployed on an Active server and a community of SOAP enabled sensors and actuators. Input sensors are speech recognition, vision based gesture recognition and probes used to monitor patient vital signs. Actuators are the main user interface, a robotic endoscope holder and a speech synthesizer.

The system is evaluated and reviewed by surgeons and medical equipment suppliers on a regular basis. For the first time, a natural and intuitive computer interface enables them to interact with computers as though they were an active member of the team. In addition, a service-based architecture federates computer based systems present in the operating to centralize all interactions through the same set of multimodal channels. It saves surgeons from learning about different system designs and limits the number of user interfaces they have to deal with.

Since the system is built as a community of distributed services, multiple surgeons can collaborate from different locations by dynamically connecting their own user interfaces on a shared network.

The major problem we see for a broader deployment of our system is the standardization of the operating room components. Operating rooms communication protocols are being developed, but they are not open and use proprietary technologies.

SUMMARY AND FUTURE WORK

The Active framework provides a unified tool and approach for rapidly developing applications incorporating robust natural language interpretation,

dialog management, multimodal fusion and brokering of web services. As such, Active aims to unleash the immense potential of intelligent software by making required technologies more easily accessible.
Its goal is foster research and innovation in this new field of software design by helping launch more academic and commercial projects. Active has been used in various domains, such as intelligent spaces and ubiquitous mobile communications.
In the medical field where computers are part of the standard equipment of surgery rooms, an Active based intelligent operating environment is under development and evaluation. This software assistant enables surgeons to interact with computer systems as if they were an active member of the team. More work remains to be done on both implementation and methodology aspects of Active. To perform realistic clinical tests, we are working on integrating real operating room components with the Active framework.
If Active has proven techniques for basic language processing and service orchestration, further investigation needs to be done on activity recognition and plan execution. Our philosophy is to use the Active framework to unify these two disciplines to perform them in a unique environment. Active could then look at the activity of a user, understand what is being attempted to proactively provide relevant assistance or take over the execution of the task.

ACKNOWLEDGEMENTS

This research has been supported by SRI International and the NCCR Co-Me of the Swiss National Science Foundation.

BIBLIOGRAPHY

[1] MAES P.,
Agents that reduce work and information overload
Communications of the ACM, 1995, 38.

[2] SOWA J.F.,
Architecures for intelligent systems. Special Issue on Arti¯cial Intelli-
gence of the IBM Systems Journal, 2002, 41 : 331-349.

[3] WINIKOFF M., PADGHAM, L. HARLAND.
Simplifying the development of intelligent agents
Australian Joint Conference on Artificial Intelligence, 2001, 557-568.

[4] MIDDLETON S.E.
Interface agents: A review of the field, 2002.

[5] MORRIS J., REE P.,MAE P.
Sardine: dynamic seller strategies in an auction marketplace
ACM Conference on Electronic Commerce. 2000, 128-134.

[6] BERRY P., MYERS K., URIBE T., YORKE-SMITH N.
Constraint solving experience with the calo project
Proceedings of CP05 Workshop on Constraint Solving under Change and Uncertainty, Sitges, Spain, 2005 4-8

[7] CHEYER A., MARTIN D.
The open agent architecture
Journal of Autonomous Agents and Multi-Agent Systems. 2001, 4(1) : 143-148.

[8] SYCARA K., DECKER K., PANNU A.S., WILLIAMSON,.M, ZENG D.
Distributed intelligent agents
IEEE Expert, 1996

[9] RAO A.S, GEORGEFF M.P.
BDI-agents: from theory to practice
Proceedings of the First Intl. Conference on Multiagent Systems, San Francisco, 1995.
[10]MYERS K..
A procedural knowledge approach to task-level control
In proceedings AIPS-96, 1996, AAAI Press 1996 158-165
[11] NORLING E., RITTER F.E.
Embodying the JACK agent architecture
Australian Joint Conference on Artificial Intelligence. 2001, 368-377.

[12] GRAETZEL C., FONG T.W, GRANGE S., BAUR, C.
A non-contact mouse for surgeon-
computer interaction
Technology and Health Care 2004, 12(3) : 245-257