ACTIVE, A PLATFORM FOR BUILDING INTELLIGENT SOFTWARE

Didier Guzzoni
Robotics Systems Lab (LSRO2)
Swiss Federal Institute of Technology (EPFL)
CH-1015, Lausanne, Switzerland
email: didier.guzzoni@epfl.ch

Adam Cheyer
Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, California 94025
adam.cheyer@sri.com

Charles Baur
Robotics Systems Lab (LSRO2)
Swiss Federal Institute of Technology (EPFL)
CH-1015, Lausanne, Switzerland
email: charles.baur@epfl.ch

ABSTRACT
Computer systems keep growing in complexity, processing power and inter-connectivity. To leverage this rich environment and better assist users, a new type of intelligent assistant software is required. Building intelligent assistants is a difficult task that requires expertise in many AI related fields including natural language interpretation, dialog management, multimodal fusion and brokering of services. We believe that providing a unified tool and a set of associated methodologies 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 in a single and coherent software environment. As an example, this paper illustrates how Active has been used to implement an intelligent assistant to help surgeons in a computer equipped operating room.

KEY WORDS
Intelligent Systems, Artificial Intelligence, Expert Systems

1 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]. In this 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 behavior.

The development of intelligent assistants requires expertise in many fields [3]. Perception of human activities is typically based on multimodal techniques involving inputs 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 dialog management are the responsibility of planning or rule based systems. Finally, as planning unfolds various actions are taken by the system. Based on their nature and purpose, intelligent systems act and communicate through a wide range of modalities. Such modalities need to be dynamically selected based on the execution context, thus requiring service and resource orchestration. For instance, to deliver a message, a pervasive intelligent assistant needs to pick the right modality (email, instant messenger, SMS) based on the location and availability of a user. Deploying intelligent assistants software is a difficult task. Due to the variety and complexity of technologies required, intelligent assistants are made of collections 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 help solve 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. Section 2 outlines the Active original concepts, architecture and current implementation. Section 3 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.

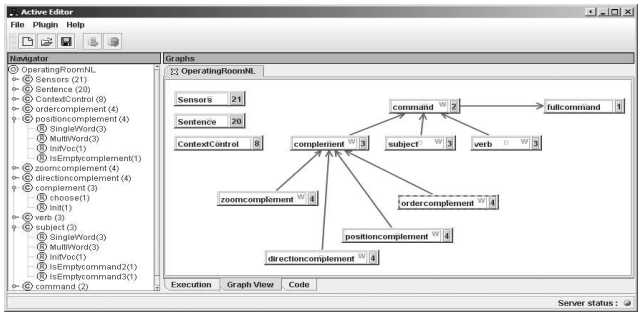

Figure 1. Language processing Active Ontology in the Active Editor

2 Related work

By definition, intelligent interactive systems are based on various AI techniques. Relevant efforts related to our research can be classified into four 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 assistants are also relevant in the domain of heterogeneous smart spaces, instrumented rooms able to sense their environment and act upon events and conditions. 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.

Secondly, our work 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. The design unifies in a single formalism the application domain knowledge, the messages exchanged among agents, the capabilities of agents and data driven events. 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 components to form communities of agents. Similarly, the Retsina [8] framework is an 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.

Intelligent behavior can also be modeled with goal oriented programming systems. 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.

Finally, rule based expert system engines such as Clips [12] or SOAR [13] have been designed to model intelligent applications. Such systems use a knowledge base, an inference engine and a set of rules to represent the behavior of a software application. Both Clips and SOAR are robust, proven and effective implementations of general purpose rule based systems. Although based on a similar rule based principles, the aim of the Active framework is to not only provide a tool, but also a set of methodologies and extensions to encapsulate the set of AI techniques required to build intelligent assistants. To support its own set of specific constraints, such as time based conditions, the current implementation of Active uses its own rule engine. Integrating Clips or SOAR into future versions of Active is under evaluation.

3 The Active framework

3.1 Conceptual overview

Our solution, the Active framework provides a unified tool and methodology to ease 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. Concepts reason about incoming channel messages, to determine how the information should be combined and acted upon. As new inputs arrive in the system, Sensor Concepts begin spreading this information throughout the network using a time-sampling method for managing high-frequency changes in a scalable fashion. Sensor observations are combined with current context, merged with other sensed information, and when enough evidence is presented to infer intent, action requests are triggered and flow through process structures modeled in the network. Throughout this progression, *meta-concepts* can reason about the overall activity flow, about time constraints, and so forth. Reasoning is modeled as rule sets attached to Concepts across an Active Ontology. Active hosts a fact base, which contains elements to represent the current state of an Active program. Processing cycles evaluate rules conditions and, if they match the current state of the fact base, associated actions are executed. This approach allows programmers to graphically define and application domain as an ontology and turn it into an execution environment by laying processing elements over it. Based on this principle, Active techniques have been developed to perform natural language interpretation, dialog management, multimodal fusion and brokering of services in a single unified framework.

3.2 Technology

The Active framework implementation is a Java based software suite designed to be extensible and open. It consists of three components, the Active Editor (figure 1), the Active Server and the Active Console. The Active Editor is a design environment used by programmers to model, deploy and test Active applications. It features a graph editor to model Active Ontologies, an integrated code text editor to work on processing rules and a debugging section. Connected to a running Active Server, the debugger is used to trigger the execution of deployed Active Ontologies and collect real time events from the Active Server for visualization and analysis. The Active Server is a scalable runtime engine that hosts and executes one or more Active programs. It can either be running as a standalone application or deployed on a J2EE compliant application server. The Active server exposes a SOAP API allowing external sensors component to report their results by remotely inserting facts into fact stores, thus triggering a evaluation cycle of Active Ontologies. Finally, the Active Console permits observation and maintenance of a running Active Server. To ensure ease of integration and extensibility, components of the Active platform communicate through web service (SOAP) interfaces. For both the Active Editor and Active Server, an open (SDK) plug-in mechanism enables researchers to package AI functionality to allow developers to apply and combine the concepts quickly and easily. A set of Active extensions is available for language parsing, multimodal fusion, dialog management and web services integration.

4 Use case : an intelligent assistant for the operating room

Based on Active technology, an intelligent operating assistant for neurosurgery has been implemented and is currently evaluated by surgeons. The system consists of a multimodal interface 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. Since computers and their peripherals are difficult to sterilize, they cannot be directly used by surgeons when operating. Therefore, interaction with the intelligent assistant takes place through a combination of hand gesture using a contact-less mouse [14] and voice recognition. Commands are issued to control a powered endoscope, navigate through pre-operative data and choose which information to show on the main display.

4.1 Active based application design

The surgery assistant application consists of a set of Active Ontologies deployed and executed on the Active Server and a community of sensors and actuators integrated as loosely coupled services (figure 2). Sensors (user interface, speech recognizer, stereo camera or any physical measuring probe) report events captured in the environment to the Active Server.

In response to incoming events, an Active Ontology in charge of natural language interpretation attempts to construct structured commands. This Active Ontology consists of concepts and relationships that define the domain of the application (figure 1). A tree like structure models valid commands. A command is made of a *subject*, a *complement* and a *verb*. A *complement* can either express a direction (up, down, left, right) or zoom (in, out) for camera controls or express control of an ordered sequence of items (last, first, next, previous) for image navigation. Once the domain has been defined with concepts and relationships, a layer of language processing is applied on it by inserting rule sets directly to the domain concepts. Active unique design allows programmers to model the domain of an application along with associated language processing components in a single unified workspace.

Once created, the domain definition tree is enhanced with two types of processing concepts: sensor concepts (leaves) and node concepts (non leaves). 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 communication 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, the *command* node in our example, create and rate a structured object made out of information coming from all their children. Selection nodes, the *complement* node of our example, pick the single best rating coming from their children. Node concepts are also part of the hierarchy and report ratings and decisions 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", "zoom", "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* selects the best rated value coming from its children and reports it to its parent. At the top of the network, the node *command* assembles values from its children to create the final command. 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 is deconstructed, following a top down scheme, to verify that each element is valid and semantically correct. At this stage, specialized rules generate suggestions, to inform the user about missing or incoherent elements. Rules can also use to context (user preferences, current dialog) to automatically correct or fill out missing fields.

To ease future use of this method, the technique has been encapsulated into a set of Active extensions. The Active Editor provides a set of wizards to help define the language processing attributes of concepts. At the end of a sequence of simple steps, the wizard automatically generates processing rules and attaches them to associated concepts.

The dialog context between the user and Active is maintained by the state of concepts. After processing the command "endoscope zoom in", concepts will generate ratings and remember them to create the current dialog context. To further control the zoom factor the user can simply say "in" or "out". Based on user preferences and the type of application, the dialog context can be reset when the subject changes or on a timeout basis when the user is not active for a given period of time. Based on surgeons feedback, the most appropriate technique consists of clearing the context whenever the subject changes. Time based techniques are more effective in information browsing environment such as interactive kiosks, but turn out to add unnecessary stress in a more critical situation such as an operating room.

Active is a testbed for multimodal applications where multiple sensors can contribute to make up a command.

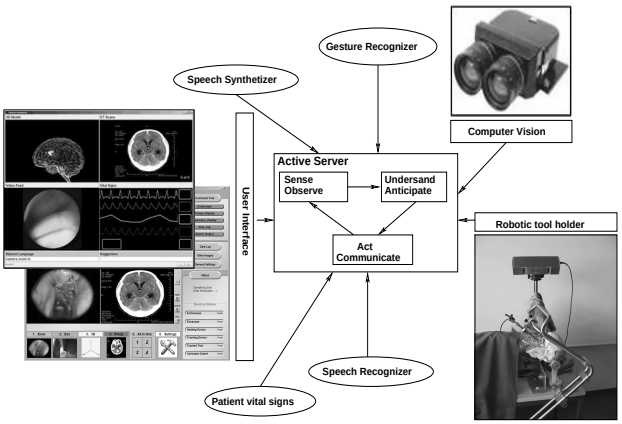

Figure 2. Active based surgery room prototype

Since sensors report events to the Active Server through a web service interface, they can be heterogeneous, distributed and easily added. For instance, a surgeon can say "move endoscope" 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 right to left. The language processing Active Ontology, using its bottom up network of concepts, will assemble these fragments to generate a full command.

Complete and valid commands are processed by a final stage, implemented as a separate Active Ontology, to perform actions through dynamic brokering of services. 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. For instance, if an important message has to be delivered to a surgeon, the media could be email or instant messaging if the user is at the computer, an alarm on a pager or multimedia message on a pocket computer for hospital personnel out of their office. This concept of delegation computing [7] is implemented by a 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.

4.2 Results

The Active based surgery assistant is reviewed by surgeons and medical equipment suppliers on a regular basis. For the first time, a natural and intuitive computer interface enables surgeons 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 room 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.

5 Summary and future work

The Active framework provides a unified tool and approach for rapidly developing applications incorporating natural language interpretation, dialog management, multimodal fusion and brokering of web services. As such, Active aims to unleash the potential of intelligent software by making required technologies more easily accessible. The Active framework implementation is a Java based software suite designed to be extensible and open. The Active Editor 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. An Active application consists of a set of Active Ontologies deployed and executed on the Active Server and a community of sensors and actuators integrated as loosely coupled services. Active is 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 has been implemented and is under 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. In the medical field, to perform realistic clinical tests, we are working on integrating real operating room components with the Active framework. As a generic software tool, we plan on further develop Active based systems on different fields of applications such as mobile and ubiquitous computing. On the methodology side, Active has proven techniques for language processing, dialog management and service orchestration. Further investigations need 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 even take over the execution of the task.

6 Acknowledgements

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

References

[1] Maes, P.: Agents that reduce work and information overload. In: Communications of the ACM. Volume 38. (1995)

[2] Sowa, J.F.: Architecures for intelligent systems. Special Issue on Artificial Intelligence of the IBM Systems Journal **41**(3) (2002) 331–349

[3] Winikoff, M., Padgham, L., Harland, J.: Simplifying the development of intelligent agents. In: 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., Maes, P.: Sardine: dynamic seller strategies in an auction marketplace. In: 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. In: 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 **4**(1) (2001) 143–148 OAA.

[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. In: Proceedings of the First Intl. Conference on Multiagent Systems, San Francisco (1995)

[10] Myers, K.L.: A procedural knowledge approach to task-level control. In Drabble, B., ed.: AIPS-96, AAAI Press (1996) 158–165

[11] Norling, E., Ritter, F.E.: Embodying the JACK agent architecture. In: Australian Joint Conference on Artificial Intelligence. (2001) 368–377

[12] Giarratano, J.C.: Clips 6.20 user's guide (2002)

[13] Laird, J.E., Newell, A., Rosenbloom, P.S.: Soar: an architecture for general intelligence. Artif. Intell. **33**(1) (1987) 1–64

[14] Graetzel, C., Fong, T.W., Grange, S., Baur, C.: A non-contact mouse for surgeon-computer interaction. Technology and Health Care **12**(3) (2004) 245–257