Active : A unified platform for building intelligent web interaction assistants

Didier Guzzoni, Charles Baur
Robotic Systems Laboratory
Swiss Federal Institute of Technology, EPFL
CH-1015 Lausanne, Switzerland
didier.guzzoni, charles.baur@epfl.ch

Adam Cheyer
Artificial Intelligence Center
SRI International
Menlo Park, CA 94025
adam.cheyer@sri.com

Abstract

Computer systems keep growing in complexity, processing power and web connectivity. To leverage this rich environment and to 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 and engineering related fields. We believe that providing a unified tool and a set of associated methodologies to create end-to-end 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 create an intelligent assistant to help mobile users retrieve online information using a multimodal dialog approach.

1 Introduction

As computer systems grow in complexity and access more local and distributed resources, they are providing rich applications that cannot be fully leveraged through classic click-and-execute interaction models. Complex applications should behave as interactive assistants to whom tasks and queries are delegated. For instance, when looking online for a local restaurant, instead of navigating through multiple web sites, it would be easier to simply send an e-mail to your personal assistant asking, in plain English, "find me a Chinese restaurant tonight in San Francisco". The answer would then come back as an email with a list of restaurants, or as a question such as "which area of San Francisco?", thus starting a computer-user dialog leading the user toward his or her goals. A wide class of problems require the capabilities of an intelligent assistant, a piece of software that can communicate with humans, understand the situation by mapping input senses into a model, act to produce useful behavior (i.e retrieving relevant information), and then interact through an appropriate rendering of communicative content.

The development of a system able to incorporate all of these capabilities requires programming in different languages and expertise in many AI-based methods, and hence is typically beyond the reach of most software developers. Such an undertaking involves a working with a collection of heterogeneous software components whose aggregate can be difficult to integrate, deploy, test and debug. Finally, combining many different approaches, tools and technologies limits the overall performance and extensibility of the system.

What if there were a toolset and an associated methodology that lowered the bar for creating intelligent applications, such that a single Java-savvy software developer could rapidly model a domain and then apply many of the best Artificial Intelligence (AI) techniques combined with web-accessible data and services through a visual, drag-and-drop interface, easy-to-use wizards, and a familiar programming language? This is the vision we are pursuing through the development of the Active framework.

This paper introduces the Active framework and presents how it is used to create an intelligent assistant able to dialog with a user to retrieve information about restaurants, movies and other points of interests from public data available on the Internet. Section 2 presents related work, section 3 introduces the Active framework, section 4 shows how Active is used to implement an intelligent assistant able to extract information from the Web through a natural language style user dialogue. Finally, a conclusion presents directions for our future work.

2 Related Work

Our research focuses on providing a unified tool to build end-to-end intelligent assistants applications. In the context of this paper, we compare our work with three fields where intelligent assistants helping users to retrieve online information have been created.

As a first category of related work, agent frameworks have been used to create personal intelligent assistants for information retrieval. In this context, the open agent architecture [4] introduces the powerful concept of delegated computing. The Retsina [7] framework is an advanced multiagent architecture to build distributed intelligent systems, offering a very effective platform to create independent reactive behavior. Finally, the Dejima system [5] offers a flexible and programmer-friendly framework for building natural language interfaces based on networks of interconnected agents. All of these agent frameworks cover parts of what is needed to create end-to-end intelligent assistants, whereas our goal is to create a more generic tool to model service brokering, reasoning, process execution, modality fusion, and language processing.

A second category proposes applications built for specific tasks [2], [3]. These systems are very effective in the domain for which they have been designed. Consisting of heterogeneous components, they nevertheless require substantial development efforts, have limited flexibility and are not easily adaptable to other fields of application.

Third, given the high potential of mobile intelligent assistants, research and industry partners are defining relevant standards. For spoken dialog, VoiceXML [6] is used to specify phone-based speech recognition applications, and in the mobile space, XHTML+Voice [1] enables voice to complement visual form interfaces. These techniques and Active have many similarities. We should nevertheless contrast the VoiceXML finite-state automata style of interaction, where you traverse fixed menus, to Active's more flexible dialog approach, where you can talk naturally about any part of the domain at any time. In addition, Active is a generic platform aimed at not only modeling dialog management, but full end-to-end intelligent applications.

3 Active Framework

Active introduces the original concept of *Active Ontologies* as the foundation for creating intelligent applications. Whereas a conventional ontology is a type of data structure, defined as a formal representation for domain knowledge with distinct classes, attributes, and relations among classes, an Active Ontology is a processing formalism where distinct processing elements are arranged according to ontology notions; it is an execution environment.

Active Ontologies are made up of a relational network of *concepts* connected by *relationships*. Concepts and relationships serve to define both data structures in the domain (e.g. a movie has a title, actors, a genre, a rating) as well as a processing environment using rule sets that perform actions within and among concepts. At the core of Active is a specialized rule engine, where data and events stored in a fact base are manipulated by rules written using JavaScript augmented by a light-layer of logic-style unification à la Prolog. JavaScript was chosen for its robustness, clean syntax, popularity in the developer community, and smooth interoperability with Java. Unification was chosen for its rich matching capabilities so often used in production rule systems.

The Active platform is implemented as Java-based software suite designed to be extensible and open. It consists of three components. The *Active Editor* (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 programs. It can either be run as a standalone application or deployed on a J2EE compliant application server. 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. An Active-based application consists of a set of loosely-coupled services working within one or more Active Ontologies.

On top of the design and implementation described in the previous section, we have been encoding a number of AI approaches using Active. Currently, Active can be used to perform language parsing, multimodal fusion, dialog and context management, and web services brokering and integration. Each methodology comes as set of Active Ontologies and wizards for rapid assembly by developers. Wizards are part of a plug-in mechanism that enables researchers to package AI functionality to allow developers to apply and combine the concepts quickly and easily.

4 Interactive Web Information Retrieval

Based on the current set of Active capabilities, it is possible to build an end-to-end intelligent application in a single unified framework. In our example, we have used Active to model and deploy a natural language driven assistant able to retrieve information about restaurants and movies. Geographically-relevant content is provided through web-service interfaces provided by Yahoo and Google.

4.1 Natural Language Processing

To create a language processing application using Active Ontologies, the first step consists of using the Active Editor to graphically specify the application domain, sketching key concepts and relationships (figure 1). In our example, the domain consists of *commands* made out of a *verb* and an *event*. An *event*, can either be a *movie event* or a *meal event*. Events have a *date* and more details about their structure. For instance, a movie has a *genre*, *actors* and a *rating*. Once the domain has been defined, a layer of language processing is applied by associating rule sets directly

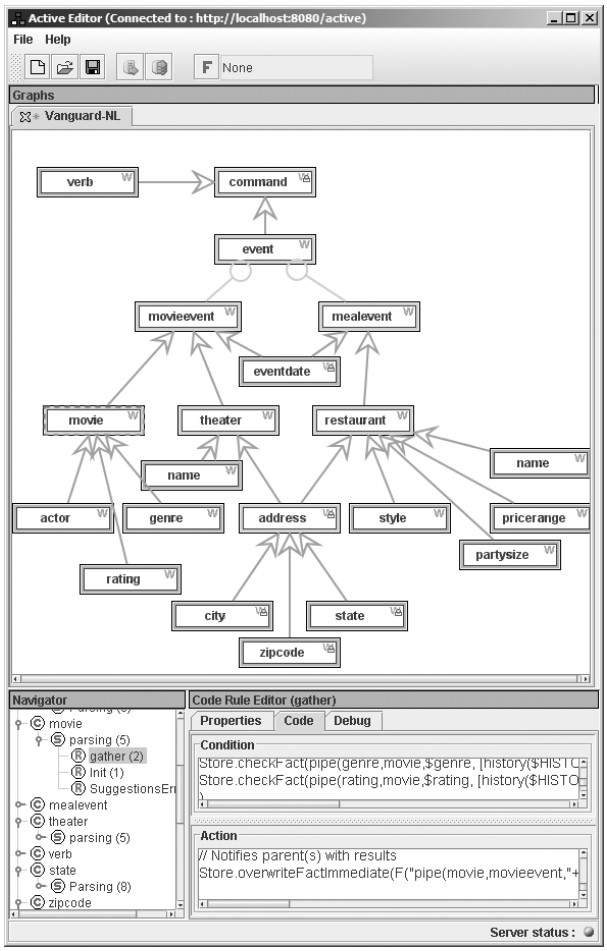

Figure 1. Active Editor

to the domain concepts. The unique design of Active allows programmers to model the domain of an application and the associated language processing component in a single unified workspace.

The ontology-like domain definition is a dynamic processing environment which reacts to incoming words (captured by the user interface) to produce a command to be performed by the system. The domain tree has two types of processing concepts: *sensor* concepts (leaves) and *node* concepts (non-leaves). Leaf concepts are specialized filters to sense and rate incoming words about their possible meaning. Typically sensor concepts generate ratings by processing the sequence of incoming words. Words are tested using regular expression pattern matching or compared against a known vocabulary set.

Sensors report their results to their node parents. There are two types of *node* concepts: *gather nodes* and *selection nodes*. Gather nodes (i.e the *movie* node) create and rate an aggregated structured object by combining inputs coming from their children. Selection nodes (i.e the *event* node) choose 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 tokens are incrementally assembled up the domain tree to produce a structured command at the root node.

The Active network of concepts holds the context of the conversation. Nodes and leaves remember their current state, allowing the user to use partial sentences, or complete information when necessary. One can say ”find movies tonight in Palo Alto” and provide a second utterance later as ”in San Francisco” to change the current location or ”Italian restaurants” to change the subject but keep the current location. This type of dialogue is well suited for mobile communication interactions where bandwidth and user interface complexity are limited.

Relationships play an important role in our approach. Two types of relationships are used: *structural* and *classification*. Structural relationships (arrow ended links on figure 1) represent structures by binding elements together, they relate to a *”has a”* ontological notion. For instance, *actor*, *genre*, *rating* and *moviename* are members of a *movie*. Structural relationships also carry cardinality information and record whether children nodes are optional or mandatory. This information is used by Active to provide the user with contextual information. For instance, if a user initiates a dialog with ”find Italian restaurants”, since the *address* node is mandatory for movies and restaurants, the user will be asked to provide an address. Through this mechanism, the parsing tree not only generates a structured command but also information to interactively assist the user. Classification relationships (circle ended links on figure 1) represent nodes types and how they should be selected, they relate to a *”is a”* ontological notion. For instance, both *movieevent* and *mealevent* are *events* and one of them will be picked upon evaluation.

The system is capable of automatically completing user commands. In our case, the *address* node has specific rules to automatically fill the *state* and *city* slots if a *zip code* is provided. It invokes an external resource (a SOAP web service) to try and retrieve city information out of a zip code. The service is invoked using a specialized Active Ontology in charge of dynamically pick and invoke web services based on criteria such as availability and current execution context. (See section 4.2)

Changes to language domain definition and processing are easily and graphically done through the Active Editor. There is no code to edit, system to shutdown nor program to recompile. To add a new attribute *rating* to a *movie*, a user uses the *Add Leaf* wizard to define parameters and rules that control how leaf nodes rate incoming words. The wizard automatically generates a new concept and its processing

rules. The next step consists of graphically connecting the new leaf with the *movie* node with a *structural* relationship and specifying its attributes (mandatory, cardinality). The final step is to redeploy the modified Active Ontology onto the Active Server.

4.2 Web Service Brokering

To connect Active based intelligent assistants with their environment, a set of wizards and specialized Active Ontologies have been designed to register, dynamically select and invoke web services. Each registered service belongs to a well defined *service category* that specifies input parameters, output parameters and selection attributes. For instance, a *restaurant information provider* category takes a zip code, a food type and price range as inputs to return a set of restaurants, each containing restaurant details and an address. Service categories are represented with Active concepts and relationships easily and graphically created using the Active Editor.

Registering service providers for a service category is interactively done using the Active *web service registration* wizard. This wizard takes the name of the service, the category where it belongs and a pointer to a valid WSDL[1] document. Based on the WSDL document, a mapping tool is provided to match the specific API of the service provider with the Active-defined category under which the service is registered. Using this technique, many service providers, each with their own unique web service interface, can be registered under the same category. A specialized Active Ontology has been developed that helps dynamically select, invoke, and coordinate services by category. This technique allows callers to delegate the specific instance of service to call by providing a category of service to invoke, along with inputs parameters and selection attributes. In our example, four service categories and providers have been defined: The Restaurant and Movie service categories are served by web services from Yahoo and Google respectively. A specialized service in charge of providing city data and zip codes is used as a *city information* provider. Finally, the *user interface* category delegates work to different types of interfaces (e.g. email, PDA, web, voice) to capture user inputs and provide them with results.

4.3 Results

The framework described here has been used to implement an end-to-end intelligent web information retrieval application. The system offers two types of user interfaces. Either a simple Java Swing based console or a mobile phone hosted application coupled with a speech recognizer allow users to initiate a dialog to retrieve online information about movies and restaurants. With help from industrial partners, we have begun to evaluate the Active approach for building intelligent and natural interaction with web information to mobile users.

[1] Web Services Description Language

5 Conclusion

The Active framework provides a unified tool and approach for rapidly developing applications incorporating natural language interpretation, dialog management and access to web information. Our future efforts focus on Active methodologies and applications: On the methodology side, we are incorporating Active-based approaches to activity recognition and time-constrained process execution. Our philosophy is to use the Active framework to unify these two disciplines, allowing a single domain model to be used when watching the activity of a user, understanding what is being attempted, and then to proactively providing relevant assistance during execution of the task. Finally on the application side, we are working with industrial partners on hardening Active's main components, improving the robustness, flexibility, usability, and scalability of the development tools and runtime platform. Acknowledgements : this research is supported by SRI International and the NCCR Co-Me of the Swiss National Science Foundation.

References

[1] J. Axelsson, C. Cross, H. Lie, G. McCobb, T. Raman, and L. Wilson. Xhtml+voice profile 1.2. W3C working draft, W3C, 2004. http://www.voicexml.org/specs/multimodal/x+v/12/.

[2] P. Berry, K. Myers, T. Uribe, and N. Yorke-Smith. Constraint solving experience with the calo project. In *Proceedings of CP05 Workshop on Constraint Solving under Change and Uncertainty*, pages 4–8, Sitges, Spain, oct 2005.

[3] M. Budzikowska, J. Chai, S. Govindappa, V. Horvath, N. Kambhatla, N. Nicolov, and W. Zadrozny. Conversational sales assistant for online shopping. In *HLT '01: Proceedings of the first international conference on Human language technology research*, pages 1–2, 2001.

[4] A. Cheyer and D. Martin. The open agent architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 4(1):143–148, March 2001. OAA.

[5] B. Hodjat, H. Franco, H. Bratt, K. Precoda, A. Stolcke, A. Venkataraman, D. Vergyri, and J. Zheng. Iterative statistical language model generation for use with an agent-oriented natural language interface. In *10th International Conference on Human-Computer Interaction*, Crete, June 2003.

[6] J. Larson. Voicexml and the w3c speech interface framework. *IEEE MultiMedia*, 10(4):91–93, 2003.

[7] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The RETSINA MAS infrastructure. Technical Report CMU-RI-TR-01-05, Robotics Institute Technical Report, Carnegie Mellon, 2001.