ACTIVE, A TOOL FOR BUILDING INTELLIGENT USER INTERFACES

Didier Guzzoni and Charles Baur
Robotics Systems Lab (LSRO 2)
EPFL
Lausanne, Switzerland

Adam Cheyer
Artificial Intelligence Center
SRI International
Menlo Park, California, USA

ABSTRACT
Computers have become affordable, small, omnipresent and are often connected to the Internet. However, despite the availability of such rich environment, user interfaces have not been adapted to fully leverage its potential. To help with complex tasks, a new type of software is needed to provide more user-centric systems that act as "intelligent assistants", able to interact naturally with human users and with the information environment. 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, combines an innovative production rule engine with communities of services to model and implement intelligent assistants. In the medical field, our approach is used to build an operating room assistant. 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 active members of the team. In a broader context, Active aims to ease the development of intelligent software by making required technologies more accessible.

KEY WORDS
Man-Machine Interfaces, Intelligent Systems, Cognitive Processes, Medicine

1 INTRODUCTION

A growing number of applications require intelligent user interfaces to whom tasks can be delegated in a natural and interactive manner [1]. Computers should be seen as personal assistants rather than rigid tools controlled through the basic "click-and-do" paradigm. To fully leverage the power of today's modern computing environment where processing power is affordable, omnipresent (mobile devices, cars, appliances) and always connected (Wifi, WiMAX) computers should be told "what to do" instead of "how to do". We define an "intelligent assistant" as a software system able to observe and sense its environment (including human communications), to analyze a situation by mapping input senses into a model of what tasks and events may be happening, and then to understand and anticipate what actions will produce relevant and useful behavior.

As an example, let's assume someone is looking for a flight from Boston to San Francisco. Instead of going over multiple web sites to get quotes, one should be able to express the request in a more natural way by, for instance, simply sending an email to an intelligent assistant saying "find me a flight from Boston to SFO next Thursday". The system would then send an email back to the user with a list of possible flights or a request for more details. Such a thread of email messages offers a natural dialog to a user interacting with an intelligent assistant.

Intelligent user interfaces are difficult to design, implement and deploy. Such software systems require expertise in many AI related fields [2]. 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 language processing, 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. On the implementation side, 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.

The goal of our research is to provide a unified tool and associated methodologies to ease the development of intelligent user interfaces. Our solution consists of a service oriented architecture where services are orchestrated by the Active system, an innovative production rule based framework. Our approach brings AI technologies to non-expert programmers, allowing them to leverage the best of AI techniques by encapsulating their underlying complexity. Programmers can model all aspects of intelligent assistant interfaces (language processing, plan execution and modality fusion) in a unified and programmer friendly

framework.

This paper presents how our approach is used to create a multimodal assistant designed to help surgeons in the operating room. The next section is dedicated to related work. Then, we outline the Active framework, its original concepts, architecture and current implementation. Next, we present in more details how our framework is used to design and implement an intelligent assistant for the operating room. Finally, a conclusion presents directions of our future work.

2 RELATED WORK

In the field of multimodal user interface framework, the open agent architecture [3] (OAA) introduces the powerful concept of delegated computing. Similarly to our approach, OAA systems consist of communities of services whose actions are combined to execute complex plans. 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 agents to form intelligent communities.

The MULTIPLATFORM testbed [4] is a generic service oriented software framework to build dialog systems. It has been used in numerous applications ranging from interactive kiosks to mobile assistants. Although is has shown robustness and effectiveness, the system lacks some of the flexibility required to support dynamic planning and runtime reconfiguration. All data structures and messages exchanged among components are defined as XML documents at design time, and cannot be easily changed on the fly. Adding new types of services requires the application to be taken offline and redesigned, whereas we are trying to provide a more dynamic environment where services and service types can easily be added to the system.

The CALO project [5] aims at designing and deploying a personal assistant that learns and helps users with complex tasks. CALO is an extremely heterogeneous system, involving components written in eleven different programming languages. CALO meets the requirements for which it was designed but is not a cognitive architecture tool to be used by non expert programmers. Similarly, the RADAR project [6] is an intelligent assistant designed to help users deal with crisis management. Its flexibility and sound design have allowed the system to be effectively deployed and tested by users. However, its complexity prevents programmers from rapidly getting up to speed without learning about implementation details and AI concepts.

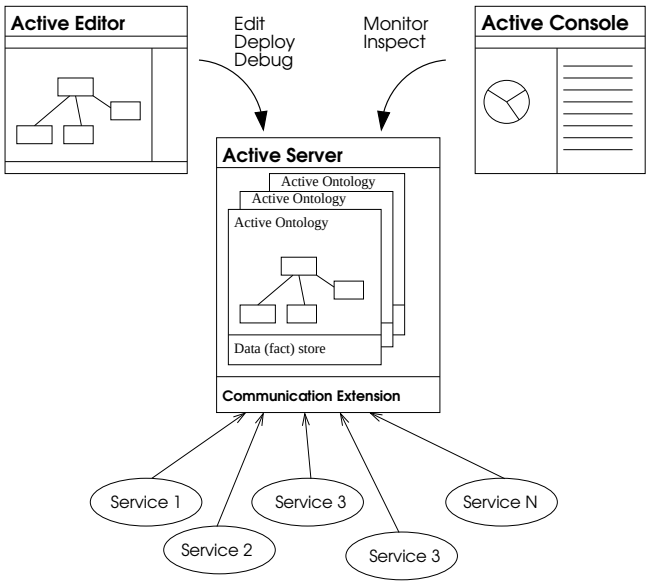

Figure 1. Active application design

3 ACTIVE FRAMEWORK

The Active system is a unified framework designed to build intelligent systems. Its goal is to lower the bar to allow more programmers to build complex intelligent interface systems featuring multimodal input, language processing, plan execution and multimodal output.

3.1 Active Ontologies

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 concepts, attributes, and relations among classes; it is a data structure. An Active ontology is an enhanced ontology where processing elements are arranged according to ontology notions; the ontology becomes an execution environment.

An Active Ontology consists of interconnected processing elements called concepts, graphically arranged to represent the domain objects, events, actions, and processes that make up an application. The logic of an Active application is represented by rule sets attached to concepts. Rule sets are collections of rules where each rule has a condition and an action. Conditions and actions are expressed in JavaScript augmented by a light-layer of firstorder logic. JavaScript was chosen for its robustness, clean syntax, popularity in the developer community, and smooth interoperability with Java. First-order logic was chosen for its rich matching capabilities (unification) so often used in production rule systems.

In addition, each Active ontology is given a data store, used to persist firstorder logic facts that represent the state and variables of the current processing. When the contents

of the fact store changes, an evaluation cycle is triggered and conditions are evaluated. Fact stores can be shared to exchange information and perform actions across Active Ontologies. Finally, stores can be accessed by external programs, so that new pieces of information can be added from the outside world to trigger further processing.

An Active-based application (see figure 1) consists of a set of loosely coupled services working with one or more Active Ontologies. Using loosely coupled services eases integration of sensors (e.g. speech recognition, vision systems, mobile or remote user interfaces), effectors (e.g. speech synthesis, user interfaces, robotics) and processing services (e.g. remote data sources, processing components).

3.2 Implementation

The current implementation of Active consists of three components. First, the *Active Editor* (see figure 2) is a design environment used by developers to model, deploy and test Active applications. Within the Active Editor, developers can graphically create and relate concept nodes, select Wizards that automatically generate rule sets within a concept to perform actions such as interpretation of natural language, modeling of executable processes, or connecting to third-party web services and finally test or modify the rule sets as needed. Second, 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. The Active server exposes SOAP or RMI apis allowing external sensors component to report their results by remotely inserting facts into fact stores, thus triggering the evaluation of concept rules within the deployed Active Ontologies. Finally, the *Active Console* permits observation and maintenance of a running Active Server.

The Active framework implementation is a Java-based software suite designed to be extensible and open. For both the Active Editor and Active Server, plug-in mechanisms enable researchers to package AI functionality to allow developers to apply and combine the concepts quickly and easily. A growing set of Active extensions is available for language parsing, multimodal fusion, dialog and context management, and web services integration. To ensure ease of integration and extensibility, all three components of the Active platform communicates through web service (SOAP) or RMI interfaces.

3.3 Methodologies

Based on the design and implementation described above, a set of Active methodologies has been created to perform language processing, dynamic service brokering and process modeling.

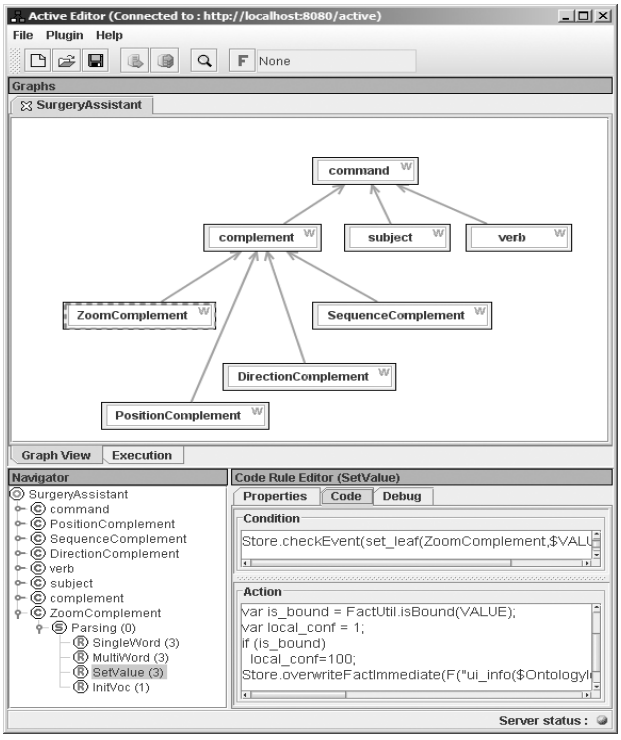

Figure 2. Active Editor

3.3.1 Language processing

The goal of a language processing component is to gather input utterances, understand their meaning, and to finally generate a command to be delegated for execution. To perform language processing, Active uses a pattern recognition technique, where ontology concepts are used to model the application domain and enhanced with a light layer of language (words and patterns). This approach is often very natural for developers, produces good results and the domain model is portable across languages.

To implement the pattern recognition approach for a domain, the first step consists of using concepts and relationships to specify the model of the application (see figure 2). A tree like structure is built, defining the structure of a valid command. In our example, a command is made of a subject, a complement and a verb. The complement can either express a direction (up, down, left, right) or zoom (in, out) for camera controls, express sequential control (last, first, next, previous) for image navigation or a position (top, bottom, left, right, front, rear) for 3D model manipulation.

Once the domain has been defined using concepts and relationships, a layer of language processing is applied, by associating rule sets directly on the domain concepts. Active's unique design allows programmers to model the domain of an application and the associated language processing component in a single unified workspace.

The domain tree has 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 the order of incoming events or checking their values using regular expression pattern matching or 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, e.g. the command node in our example, create and rate a structured object made of ratings coming from their children. Selection nodes, e.g. the complement node in our example, 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. This method has been encapsulated into a set of Active extensions and wizards.

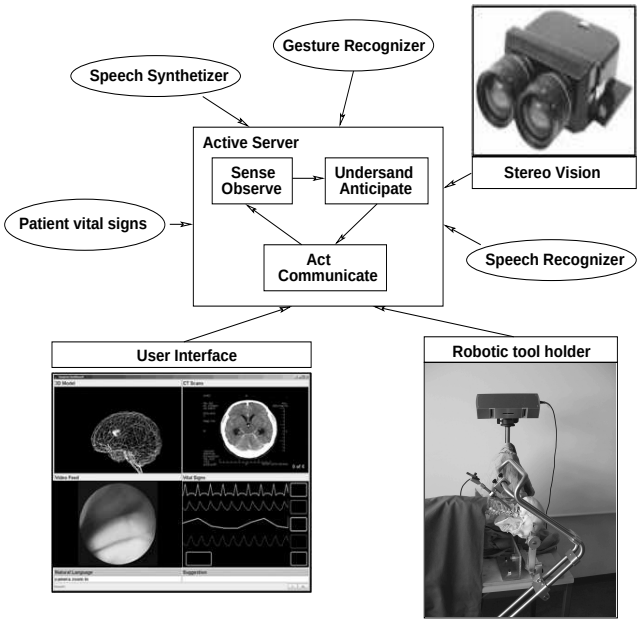

Figure 3. Active based surgery room prototype

3.3.2 Dynamic service brokering

At the heart of many multi-agent systems, such as SRI's Open Agent Architecture (OAA) [3] or CMU's Retsina [7], is a dynamic service broker which reasons about how to deal with situations where multiple service providers expose the same function. In such systems, a brokering mechanism is used to select relevant providers and gather their results on behalf of the caller. Service providers are chosen on the fly based on a service class and a set of selection attributes, which typically include properties such as service availability, user preferences, quality of service, or cost.

To implement this technique, we have created a specialized Active Ontology to work as a service registry and dynamic service broker. Service providers register their capabilities and attributes by asserting a set of fact into the associated fact store. This data set represents a simple service registry where providers can register, be discovered and invoked.

At runtime, the broker will use this information to select which providers can be called based on the caller's attributes and current context. Once a list of suitable providers have been selected, the broker invokes them using one of two techniques. First, a sequential approach, where providers are called in sequence, until one of them successfully responds. This would for instance be used to send a notification message to a user. If several service providers can send email, the message should be delivered only once. Secondly, a parallel technique where providers are concurrently invoked, their responses being aggregated into a result set. This technique is used when a caller needs to retrieve information from multiple sources.

3.3.3 Process modeling and execution

An Active methodology to model processes has been designed and implemented. Using concepts and rules it is possible to model generic processes, to use the Active environment as a business process engine. Such processes have been designed to model dialogs and sequences of actions to be undertaken by Active. As other Active methodologies, this technique has been encapsulated into a set of interactive Active Editor wizards allowing programmers to model complex processes without writing any Active code.

The execution state of processes and their instance variables are persisted as Active facts in Active data stores. A collection of functional building blocks are available to model complex processes. *Start* elements define entry points that will trigger the execution of a process. *End* elements define the end of a process execution. They clean up all the process instance related information. *Fork* and *join* elements allow to model branches (sub processes to be executed in parallel) and join them later. *Execution* nodes contain Javascript code to be executed when the interpreted of the process reaches a specific stage. *Wait* nodes have a condition based on the content of the Active fact store. Whenever the condition is valid, the flow will resume its activity. A timeout can be specified to undertake action when an awaited event does not occur.

4 THE INTELLIGENT OPERATING ROOM

Modern operating rooms are equipped with various computer systems, allowing surgeons to perform complex operations and develop new techniques to improve results, limit

the trauma of surgery on patients and shorten hospital stays. The operating room has obvious and strict constrains about space and sterilization, thus preventing the use of classic keyboards and mice. In addition, surgeons and their staff wear cumbersome outfits and always need to focus on the operating field, therefore they cannot afford to switch attention or drop their tools to interact with computer systems. According to surgeons, computers will be more effective and easily accepted if they can be seen as any other member of the team. This implies that computer-human interaction should be as natural as possible.

Our approach to implement an intelligent assistant for the operating room is to create a service oriented system (see figure 3) featuring a community of independent services orchestrated by a core application implemented as a set of Active Ontologies. The system, implemented as a multimodal interface, allows 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. Following sub sections describe the application components in more details.

4.1 Core Application

The core of the application is based on three Active Ontologies running on the Active server. They implement the behavior of the intelligent interface: language processing, plan execution and interaction with the environment. A community of loosely coupled services makes up the rest of the application by sensing the environment (speech and gesture recognizers, stereo camera, user interface) and acting (user interface, speech synthesis and optionally a robotic arm).

When a sensor gathers a piece of information from the environment, it reports it by asserting a fact into the data store of the language parsing Active Ontology. This event triggers the evaluation of running Active Ontologies that will generate the most appropriate action to perform what the user asked. Note that the system is not only aware of the surgeon's activities, but also gathers information about the condition of the patient and the status of various devices running in the operating room. It aggregates this information in its global behavior, to for instance, warn the surgeon when the patient's condition changes. As more components get integrated, the Active based surgery assistant has the potential to transform the operating room into a smart intelligent space.

4.2 User interface

The main user interface is used by surgeons to access the information they need to visualize through four main areas. Live images delivered by the endoscope, pre-operative images and 3D model representing specific patient's data and general information about the general condition of the patient.

Even if the user interface is the only component with which the user is interacting, it is only the tip of the iceberg. The user interface is a service in the community working for the user. In addition, it is possible to start a second user interface to join the community and, with no significant development effort, allow two surgeons to collaborate by sharing the same environment.

4.3 Gesture recognition

Since surgeons cannot use any mouse nor keyboard while operating, we provide them with a virtual mouse pointer by tracking their hands motion. Based on the motion information, surgeons can either use their hand directly as a mouse or perform simple gestures to perform actions.

Two motion capture techniques have been integrated into the system. First, a stereo camera [8] is used to track the surgeon's hands and feed the gesture recognizer. This technique is non intrusive, easy to install but is rather sensitive to light conditions and its accuracy is limited. Secondly, we used a method where markers are mounted on the surgeon's tool and being tracked, using pulsed infrared light, by a base station that computes their location in space [9]. This technique is more intrusive (instruments have to be equipped with markers) but provides a better precision and is less sensitive to light conditions. Thanks to our service oriented approach, both mechanism can be easily swapped without adjusting any code nor configuration parameters.

For effective and fast gesture recognition, we extended the well established libstroke 2D recognition technique [10] to work as a 3D gesture recognizer (see figure 4). LibStroke takes a stroke (set of captured positions) and converts it into a command by generating signatures. The algorithm creates a bounding box around the stroke and divides it into a 3x3 grid where each sub area is uniquely identified (1 to 9). Then, each element of the stroke is visited to find out the subarea of the matrix where it belongs. Identifications of each visited subarea are concatenated to create the signature of the stroke. The signature can then be compared to a vocabulary that binds commands to signatures. Since we are using 3D gesture capture techniques, we extended the libstroke technique to work in 3D. Instead of using a 3x3 matrix, we work with a 3x3x3 matrix consisting of 27 sub areas.

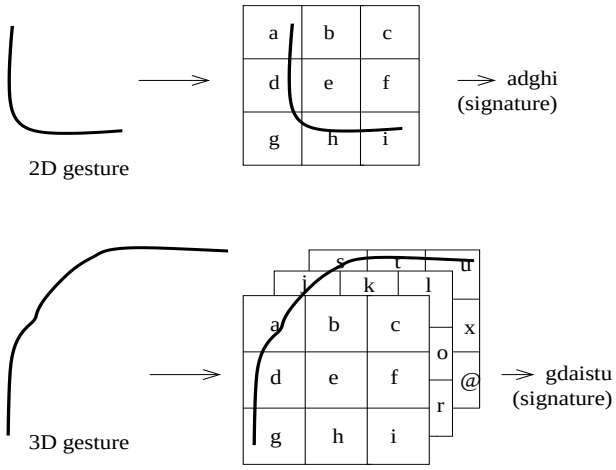

Figure 4. Fast 3D gesture recognition

4.4 Speech and sounds

Speech recognition and speech synthesis are based on the Microsoft speech SDK. In the context of the operating room, speech synthesis is not well accepted by surgeons. They would rather opt for a collection of beeps or short sounds to inform them about the status of the interface. For instance, when the system is ready for speech or gesture recognition, it emits a short sound inviting the user to speak or start a hand gesture.

5 CONCLUSION

In this paper we present an innovative architecture to develop intelligent assistants. The Active framework provides a unified tool and approach for rapidly developing applications incorporating 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. This paper shows how an Active based assistant for the operating room has been designed and successfully implemented. The current system is under review and evaluation by surgeons.

More work remains to be done on applications, implementation and methodology aspects of Active. First, on the application side, to perform realistic clinical tests of the surgery assistant, we are working on integrating real operating room components with the Active framework. In a different domain, Active is used as the back bone of a mobile assistant application that helps mobile users access data and services through a natural email based dialog. The Active framework is used in both fields, helping us improve and verify the agility and robustness of our approach. On the implementation side, we are working on scalability and robustness of the Active Server. We are planning on building clusters of Active Servers, able to balance large workloads to host multiple personal assistants serving a large number of users. Finally, we are exploring innovative AI techniques for activity representation and recognition. Our goal is to unify plan execution and activity recognition, so that an Active powered assistant could look at the activities of a user, understand what is being attempted to proactively provide relevant assistance and even take over the execution of the task.

6 ACKNOWLEDGMENTS

This research is 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] Winikoff, M., Padgham, L., Harland, J.: Simplifying the development of intelligent agents. In: Australian Joint Conference on Artificial Intelligence. (2001) 557–568

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

[4] Gerd, D.S.: (Multiplatform testbed: An integration platform for multimodal)

[5] 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

[6] Modi, P., Veloso, M., Smith, S., Oh, J.: Cmradar: A personal assistant agent for calendar management (2004)

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

[8] Graetzel, C., Fong, T., Grange, S., Baur, C.: A Non-Contact Mouse for Surgeon-Computer Interaction. Technology and Health Care **12**(3) (2004)

[9] Marti, G., Bettschart, V., Billiard, J., Baur, C.: Hybrid method for both calibration and registration of an endoscope with an active optical tracker. CARS 2004 **10**(4) (2004) 159–164

[10] Willey, M.: Design and implementation of a stroke interface library. (Technical report) http://www.etla.net/libstroke/.