GALAXY-II: A REFERENCE ARCHITECTURE FOR CONVERSATIONAL SYSTEM DEVELOPMENT[1]

Stephanie Seneff, Ed Hurley, Raymond Lau, Christine Pao, Philipp Schmid, and Victor Zue

Spoken Language Systems Group
Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139 USA

ABSTRACT

GALAXY is a client-server architecture for accessing on-line information using spoken dialogue that we introduced at ICSLP-94. It has served as the testbed for developing human language technologies for our group for several years. Recently, we have initiated a significant redesign of the GALAXY architecture to make it easier for many researchers to develop their own applications, using either exclusively their own servers or intermixing them with servers developed by others. This redesign was done in part due to the fact that GALAXY has been designated as the first reference architecture for the new DARPA Communicator Program. The purpose of this paper is to document the changes to GALAXY that led to this first reference architecture, which makes use of a scripting language for flow control to provide flexible interaction among the servers, and a set of libraries to support rapid prototyping of new servers. We describe the new reference architecture in some detail, and report on the current status of its development.

1. INTRODUCTION

In 1994, we introduced GALAXY, a client-server architecture for accessing on-line information using spoken dialogue [1]. Since then, GALAXY has served as the testbed for our research and development of human language technologies, resulting in systems in different domains (e.g., automobile classified ads [2], restaurant guide [3] and weather information [4]), different languages [5], and different access mechanisms [2, 3, 4]. In 1996, we made our first significant architectural redesign to permit universal access via any web browser [6]. The resulting WEB-GALAXY architecture makes use of a "hub" to mediate between a Java GUI client and various compute and domain servers, dispatching messages among the various servers and maintaining a log of server activities and outputs.

In the process of developing dialogue modules for various domains in GALAXY, we came to the realization that it is critical to be able to allow researchers to easily visualize program flow through the dialogue, and to flexibly manipulate the decision-making process at the highest level. To this end, we developed a simple high-level scripting language that permits boolean and arithmetic tests on variables for decisions on the execution of particular functions. We found this mechanism to be very powerful, and were successful in incorporating it into our newest domain servers for weather and flight status information. We then began to contemplate the idea of incorporating an analogous mechanism into the program control of the entire system, which was being maintained by the GALAXY hub. At about the same time, discussions were beginning on the possibility that GALAXY be designated as the reference architecture for the soon-to-be-launched DARPA Communicator Program[2] whose goal is partly to promote resource sharing and plug-and-play interoperability across multiple sites for the research and development of dialogue-based systems. It seemed possible for a scripting language, modelled after the dialogue tools developed for our domain servers, to support a *programmable* hub for the DARPA Communicator.

In the remainder of this paper, we will first discuss the design considerations. This will be followed by a description of the first implementation of this reference architecture, ending with a report on the status of its development.

2. DESIGN CONSIDERATIONS

During the design phase of GALAXY-II, several meetings were arranged, under the auspices of DARPA, among researchers from various institutions in the U.S.A. who had expertise in dialogue system architecture design. During these meetings, two topics which received a lot of attention were the nature of the control strategy and the appropriate decomposition of the server population (recognizer, dialogue control, etc.).

There appeared to be two main camps on control strategy: those who favored a more programmatic style of control (i.e., sequential rules) [7] and those who favored an open-agent architecture [8], involving either explicit or implicit message passing among the various servers. We eventually settled upon a compromise scheme which we believe will be workable for both styles of control. Our GALAXY-II implementation is based on sequential rules. We hope that at a future time someone will attempt an open-agent or message-passing implementation based on the framework described here. We believe it should require only minor extensions to the code to accommodate these needs.

In terms of the server responsibilities, it was logical to define separate servers for speech recognition, natural language understanding (which was renamed as "frame construction" to minimize overgeneralization of terminology), natural language generation, and speech synthesis. However, the components that

[1]This research was supported by DARPA under contract N66001-96-C-852, monitored through Naval Command, Control and Ocean Surveillance Center.

[2]For a description of the Communicator Program and documentation of the architecture, see http://fofoca.mitre.org.

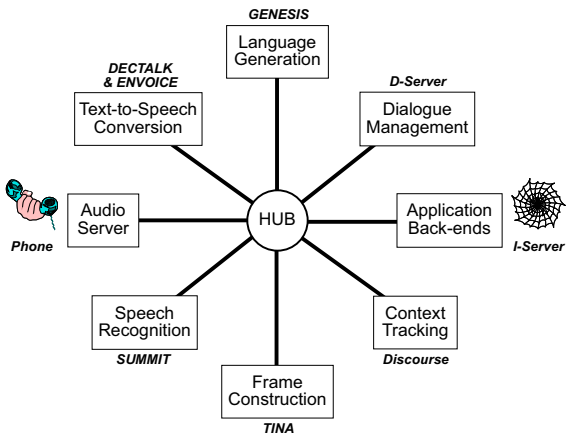

Figure 1: Architecture of GALAXY-II.

dealt with discourse, dialogue, and database retrieval were not sorted out in the same way among the groups represented at the meetings. Our group at MIT did, however, acquire from the discussions a clear vision of how we should reorganize various systems based on the GALAXY architecture to "comply" with the constraints of the emerging DARPA Communicator standards.

The resulting configuration of the GALAXY-II architecture is shown in Figure 1. The new architecture retains the client-server nature of the original design. The boxes in this figure represent various human language technology servers as well as information and domain servers. They are labelled with their associated roles as agreed upon by the committee. The label in italics next to each box identifies the corresponding MIT system component. The main organizational changes from the previous version of GALAXY were to force our domain servers to redirect any former communications with our generation system, GENESIS [11], and with any database servers (I-servers in our terminology), such that, in the new configuration, these interactions were mediated by the hub and managed in the hub script. The other major organizational change was to extract the user interface functionality into separate audio and GUI servers, further simplifying the internal responsibilities of the hub. In addition, we regularized the communication protocols for all of the servers so that they could share a common server shell library.

3. IMPLEMENTATION

The hub interaction with the servers is controlled via a scripting language. A script includes a list of the *servers*, specifying the host, port, and set of operations each server supports, as well as a set of one or more *programs*. Each program consists of a set of *rules*, where each rule specifies an *operation*, a set of *conditions* under which that rule should "fire," a list of input and output *variables* for the rule, as well as optional store/retrieve variables that come from the discourse history. When a rule fires, the input variables are packaged into a *token* and sent to the server that handles the operation. The hub expects the server to return a token containing the output variables at a later time. There is the option of no output variables, in which case interaction is one-way only. The input and output variables are all recorded in a hub-internal *master token*. The discourse history will also

```
RULE:      :ParseFrame & !:RequestFrame → context_tracking
RETRIEVE:  :HistoryFrame
IN:        :ParseFrame
OUT:       :RequestFrame :HistoryFrame :Domain
STORE:     :HistoryFrame
```

Figure 2: Example rule in the hub script.

be updated, if the rule has so specified. The conditions consist of simple logical or arithmetic tests on the values of the typed variables in the master token. The hub communicates with the various servers via a standardized frame-based protocol.

A particular *dialogue session* is initiated by a user either through interaction with a graphical interface at a Web site, through direct telephone dialup, or through a desktop agent. Our current plan is to support graphical and/or spoken interactions with multiple users simultaneously via a single hub instantiation, with the record of the current status of each session (its input/output language(s), current domain, dialogue history, etc.) being separately maintained in the hub. Tokens from multiple simultaneous sessions would compete for the server resources as specified by the hub script.

3.1. Example Rule

An example rule is shown in Figure 2. This rule states that, if a ParseFrame exists but a RequestFrame has not yet been generated (denoted by "!"), then call the "context_tracking" operation, by sending the ParseFrame to the discourse module for evaluation. Also send the previous history to define the context, which is retrieved from the hub's history record logged with the session. This operation, when completed, will return a token to the hub, containing an updated version of the HistoryFrame, a commitment to a particular domain, and a RequestFrame, which is the user query expanded to include any augmentations due to inheritance rules. The updated HistoryFrame is stored in the history record to become context for the next utterance.

3.2. Program Flow Control

A simple communication protocol has been adopted and standardized for all hub/server interactions. Upon initiation, the hub first handshakes with all of the specified servers, confirming that they are up and running and sending them a "welcome" token that may contain some initialization information, as specified in the hub script. The hub then launches a wait loop in which the servers are continuously polled for any "return" tokens[3]. Each token is named according to its corresponding program in the hub script, and may also contain a rule index to locate its place in program execution[4], and a "token id" to associate it with the appropriate master token in the hub's internal memory. The rule is consulted to determine which "OUT" variables to update in the master, and which variables, if any, to store in the discourse history. Following this, the master token is evaluated against the complete set of rules subsequent to the rule index, and any rules that pass test conditions are then executed. In the current

[3]Servers can also spontaneously send tokens to the hub without having first received a token from the hub.

[4]The sequential constraints can be easily eliminated, leading to a more open-agent architecture type of control.

implementation, the usual case is that only one rule fires, although simultaneous rule executions can be used to implement parallelism, a feature that we have exploited, for example for N-best processing. Servers other than those that implement user-interface functions are typically stateless; any history they may need is sent back to the hub for safekeeping, where it is associated with the current utterance. Common state can thus be shared among multiple servers.

To execute a given rule, a new token is created from the master token, containing only the subset of variables specified in the "IN" variables for the rule in question. This token is then sent to the server assigned to the execution of the operation specified by the rule. If it is determined that the designated server is busy (has not yet replied to a preceding rule either within this dialogue or in a competing dialogue) then the token is queued up for later transmission. Thus the hub is in theory never stalled waiting for a server to receive a token. The hub then checks whether the server that sent the token has any tokens in its input queue. If so, it will pop the queue before returning to the wait loop.

Thus far we have only experimented with N-best lists as outputs from the recognizer (as opposed to word graphs, for example). The recognizer sends to the hub each hypothesis as a separate token, signalling completion with a special final token. All of the tokens for a given utterance are jointly considered, after frame construction, by a "gather" server (omitted from Figure 1 for simplicity), which takes into consideration the quality of each hypothesis' meaning analysis, specially selecting for any salient words (cities in focus in Jupiter, for example) from the dialogue context. Early decisions are made when a hypothesis is perfect; otherwise the final decision is delayed until the last hypothesis has been processed. The gather server would also be responsible for rejecting an utterance if the system judges all hypotheses to be implausible [**?**].

The selected token is processed through discourse inheritance via the hub script and sent on to the dialogue manager. The dialogue manager usually initiates a subdialogue in order to retrieve information from the database. It composes a frame which is sent to GENESIS for paraphrasing into SQL, and the output string is then passed along to the database for retrieval. The retrieved database entries are returned to the dialogue manager for interpretation. These activities are controlled in a separate program in the hub script, which we refer to as a module-to-module subdialogue. Finally, the dialogue manager sends a reply frame to the hub which is passed along to generation and synthesis. After the synthesized speech has been transmitted to the user, the audio server is freed up to begin listening for the next user utterance.

3.3. System Development Tools

The hub can be run in a debugging mode in which the hub script can be stepped through one rule at a time. The hub script also controls specification of the variables to be recorded in a log file, another useful debugging device. Spoken waveforms are automatically recorded to file for later system training.

The hub contains a special internal "server," which handles meta-level commands that manipulate its history record, such as "scratch that" and "clear history." This server also provides session management and asynchronous i/o support functions.

Each server is compiled on top of a *server shell* library, which provides convenient routines for interfacing with the hub, as well as a support mechanism for a top-level scripting language to manage program flow, as described in the introduction. This language is similar to the one used by the hub, but has a simpler protocol. This mechanism has been used thus far only by our database and domain servers for managing dialogue control, but we expect to incorporate it into the NL servers as well.

Most but not all of the communications among the various servers are routed through the hub. However, high data-rate messages (e.g., speech waveforms) are *brokered* by the hub instead, in order to reduce the network load. The audio server sends the hub a token informing it of an impending waveform. The hub consults the existing recognizer servers, as dictated by the hub script, to determine if any are free to receive a waveform at this time, and then directs the free recognizer to receive the waveform data directly from the audio server. We envision that M recognizers would be jointly servicing up to N simultaneous conversations ($M < N$) with resource sharing taking place on an utterance-by-utterance basis, as contrasted with assigning a recognizer for the duration of a dialogue, which would be less efficient.

4. SEMANTIC FRAME REPRESENTATION

We expect that researchers utilizing the GALAXY-II system will be developing servers which will need to interface with a suite of existing servers already in place. In such cases, it is necessary for the servers to share a common language in the representations they jointly process. Researchers who choose to replace *all* the servers are free to use whatever meaning representations they find convenient. However, if the intent were to replace a subset of servers, for example, a new dialogue manager or a new language generation server, then the new server would have to adopt the meaning representation protocol that was in use by the replaced component. Thus we think it is appropriate to provide a brief description of the meaning representation formats that have been adopted by our systems.

In the process of developing conversational systems in multiple domains over the last decade, we have constructed a minimal linguistic specification of a meaning representation that we feel is adequate for most applications of interest to us. Our TINA system [9] outputs *semantic frames* in this format, and our GENESIS [11] system can paraphrase in multiple languages from this representation. Our discourse component [12] depends critically upon this format for proper functioning, and our domain servers [2, 3, 4] construct reply frames in this format, which are used as input to our ENVOICE synthesizer [13].

We view the linguistic/semantic world as consisting of three main types of constituents, which we call *clause*, *topic*, and *predicate*[5]. A clause constituent generally occurs at the highest level, and usually represents the high level goal of the user request, which could be, for example, "display," "record," "repeat," "reserve," etc. Topics generally correspond to noun phrases, and predicates are typically attributes, which could be expressed as verb phrases, prepositional phrases, or adjective phrases. A semantic frame is, then, a named and typed structure, with one of

[5] A somewhat different intent from the usual definitions of these terms.

```
clause:
  { display
    topic:
        { flight
          number: pl
          predicate:
               { from
                 topic: { city name: Boston }
               }
          predicate:
               { to
                 topic: { city name: Denver }
               }
         }
  }
```

Figure 3: Example semantic frame for the sentence, "Show me flights from Boston to Denver."

the above three types.

Semantic frames also contain *contents*, and there is a library of tools available for manipulating the contents. Traditional linguistic contents include an optional topic and zero or more predicates. A frame can also contain a set of (key: value) pairs, where the key can be any symbol-string, and the value is one of: (1) an integer, (2) a string, (3) a semantic frame, and (4) a list of values in categories (1)-(3). We use the (key: value) notation for syntactic features such as number and quantification. Conjunction is encoded with a "conj" key, and there is a distinguished "name" key for named entities. The (key: value) notation is very generic, however, and it has allowed us to represent almost any information we need to record, for example database retrievals, in semantic frame format. For instance, the key "airline" has the value "United" as retrieved from the database. In fact, the token that is sent between the hub and the servers is also itself an instance of a semantic frame[6], although at the highest level it only utilizes the (key: value) feature of the frame.

An example semantic frame for the sentence "Show me flights from Boston to Denver." is shown in Figure 3.

5. STATUS

The redesign and implementation of the GALAXY architecture started in early January 1998. At this writing, the programmable hub is fully functional, and has been delivered to MITRE[7], who will serve as the custodian for its maintenance and distribution. It includes a graphical interface accessible via a browser and an audio interface accessible by either a microphone or a telephone. An initial implementation of a complete JUPITER system [14] has also been implemented and delivered to MITRE. This system has been extensively tested, both through batch-mode regression testing and actual telephone deployment. Interested parties are referred to the MITRE web site (http://fofoca.mitre.org) for further information regarding the architecture, the API, documentation, and procedures for acquiring it.

[6]It is a clause whose name is the program it refers to.

[7]Beta-releases have also been distributed to a few other organizations including Intel, Lockheed-Martin, and HRL.

6. ACKNOWLEDGMENTS

The work described in this paper encompasses the efforts of many other present and past members of the Spoken Language Systems Group. They include: Giovanni Flammia, Jim Glass, Dave Goddeau, Lee Hetherington, Joe Polifroni, and Jon Yi. The redesign of GALAXY has benefitted from discussions with researchers at AT&T (Esther Levin and Roberto Pieracinni) Microsoft (Xue-Dong Huang), MITRE (Sam Bayer, Lynette Hirschman, Susann Luperfoy, and Rod Holland), OGI (Phil Cohen) and University of Rochester (James Allen).

7. REFERENCES

1. P. Cohen, A. Cheyer, M. Wang, and S.C. Baeg. "An Open Agent Architecture," *Proc. AAAI Spring Symposium*, pp. 1–8, Mar. 1994.
2. J. Glass, J. Polifroni and S. Seneff, "Multilingual Language Generation Across Multiple Domains," *Proc. ICSLP '94*, pp. 983–986, Yokohama, Japan, Sept. 1994.
3. J. Glass and T.J. Hazen, "Telephone-based Conversational Speech Recognition in the Jupiter Domain," *These proceedings*.
4. D. Goddeau, E. Brill, J. Glass, C. Pao, M. Phillips, J. Polifroni, S¿ Seneff, and V. Zue, "GALAXY: A Human Language Interface to Online Travel Information," *Proc. ICSLP '94*, pp. 707–710, Yokohama, Japan, Sept. 1994.
5. C. Wang, J. Glass, H. Meng, J. Polifroni, S. Seneff, and V. Zue, "YINHE: A Mandarin Chinese Version of the GALAXY System," *Proc. EUROSPEECH-97*, pp. 351–354, Rhodes, Greece, Sept. 1997.
6. R. Lau, G. Flammia, C. Pao, and V. Zue, "WEBGALAXY - Integrating Spoken Language and Hypertext Navigation," *Proc. EUROSPEECH-97*, pp. 883–886, Rhodes, Greece, Sept. 1997.
7. H. Meng, S. Busayapongchai, J. Glass, D. Goddeau, L. Hetherington, E. Hurley, C. Pao. J. Polifroni, S. Seneff, and V. Zue, "WHEELS: A Conversational System in the Automobile Classifieds Domain," *Proc. ICSLP '96*, Philadelphia, PA, pp. 542-545, Oct. 1996.
8. C. Pao, P. Schmid, and J. Glass, "Confidence Scoring for Speech Understanding Systems," *These Proceedings*, Syndey, Australia, Nov. 1998.
9. R. Pieraccini, E. Levin, and W. Eckert, "AMICA: The AT&T Mixed Initiative Conversational Architecture," *Proc. EUROSPEECH '97* pp. 1875-1878, Rhodes, Greece, Sept. 1997.
10. S. Seneff, "TINA: A Natural Language System for Spoken Language Applications," *Computational Linguistics*, Vol. 18, No. 1, pp. 61–86, 1992.
11. S. Seneff, "Robust Parsing for Spoken Language System," *Proc. ICASSP '92*, pp. 189–192, San Francisco, CA, 1992.
12. S. Seneff, D. Goddeau, C. Pao, and J. Polifroni, "Multimodal Discourse Modelling in a Multi-User Multi-Domain Environment," *Proc. ICSLP '96*, pp. 192–195, Philadelphia, PA, Oct. 1996.
13. S. Seneff and J. Polifroni, "A New Restaurant Guide Conversational System: Issues in Rapid Prototyping for Specialized Domains," *Proc. ICSLP '96*, pp. 665-668, Philadelphia, PA, Oct. 1996.
14. J. R. Yi and J. R. Glass, "Natural-sounding Speech Synthesis using Variable-length Units," *These Proceedings.*, Sydney, Australia, Nov. 1998.
15. V. Zue, S. Seneff, J. Glass, L. Hetherington, E. Hurley, H. Meng, C. Pao, J. Polifroni, R. Schloming, and P. Schmid, "From Interface to Content: Translingual Access and Delivery of On-Line Information," *Proc. EUROSPEECH '97*, pp. 2227–2230, Rhodes, Greece, Sept. 1997.