

# ORGANIZATION, COMMUNICATION, AND CONTROL IN THE GALAXY-II CONVERSATIONAL SYSTEM<sup>1</sup>

Stephanie Seneff, Raymond Lau, and Joseph Polifroni

Spoken Language Systems Group, Laboratory for Computer Science  
Massachusetts Institute of Technology, Cambridge, MA 02139 USA  
<http://www.sls.lcs.mit.edu>

## ABSTRACT

GALAXY-II is the designated initial common architecture for the DARPA Communicator project in the U.S. Its key feature is the ability to control system integration via a run-time executable scripting language. This paper describes our experience in developing complex systems based on the GALAXY-II framework. Our current system consists of four domains and two languages (English and Mandarin). Users can interface with the system in both displayful and displayless modes. Users can switch freely among the various domains in a single conversation, and multiple users can access the system in simultaneous conversations. In addition to the hub script that controls live interaction with users, we have also configured many other hub scripts that permit various batchmode runs, including the capability to reprocess log files through improved versions of the system to measure progress. The hub scripting capability has greatly accelerated our pace of system development, and has allowed us to configure considerably more complex systems than we would have previously envisioned.

## 1. INTRODUCTION

Through our experience over the last decade in designing conversational systems, we have come to realize that an essential element in being able to rapidly configure new systems is to allow as many aspects of the system design as possible to be specifiable without modifying source code. To this end, we recently redesigned our core architecture to support complex system configurations controlled by a run-time executable scripting language. Using this new framework, we have been able to configure multi-modal, multi-domain, multi-user, and multilingual systems with much less effort than previously. We are discovering that we can now configure systems whose capabilities are well beyond what was previously considered feasible.

The resulting new architecture, GALAXY-II, introduced in [1], has been designated as the initial common architecture for the multi-site DARPA Communicator project in the United States. It differs from its predecessors mainly in two ways: (1) a central hub handles all communications among the various servers, and (2) system control flow is maintained through a specialized run-time executable programming language interpreted by the hub.

Whereas [1] described GALAXY-II as an architecture, this paper concerns our recent system development efforts, which utilize GALAXY-II to carry out complex interactions with users, with databases, and among a suite of specialized servers. We have implemented a system which

can field questions, either typed or spoken, in the following four domains: Jupiter (world-wide weather information), Pegasus (flight departure/arrival/gate information for major cities in the U.S), Mercury (flight browsing and booking), and Voyager (navigation assistance and traffic status in Boston).<sup>2</sup> The user can migrate freely among the domains in typed mode, but must explicitly initiate a domain switch in spoken mode. The user can also refer verbally to items selected in the Web graphical interface via simultaneous mouse clicks. Multiple users can interact with the system simultaneously via GUI and/or telephone interfaces. All domains have thus far been implemented in English. Mandarin and Japanese versions are under active development.

An interesting research issue addressed here is how the complex tasks of mixed-initiative dialogue systems should be partitioned into a set of semi-autonomous servers, each of which has clearly assigned roles. A single database server manages database needs for all of the domains, whereas we have thus far partitioned the task of “turn management” into separate specialized servers for each domain. The turn managers routinely consult the database multiple times in the course of resolving a single user query. Discourse inheritance is managed separately from turn management, and the context record is updated after both the user turn and the system turn.

This paper is organized as follows. In the next section, we will give an overall description of program control flow through a typical dialogue turn. In Section 3, we will describe several examples of special capabilities which can be implemented within the framework of the scripting language. Section 4 will address the issue of evaluation, and discusses how hub scripts can be configured to reprocess log files obtained at the time of the original dialogue. We conclude with a discussion of future plans.

## 2. PROGRAM CONTROL

The GALAXY system consists of a central hub that controls the flow of information among a suite of servers, which may be running on the same machine or at remote locations. The hub interaction with the servers is controlled via a scripting language. A script includes a list of the active *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 into/from the discourse history. When a rule fires, the input variables are pack-

<sup>1</sup>This research was supported by DARPA under contract N66001-96-C-8526, monitored through Naval Command, Control, and Ocean Surveillance Center.

<sup>2</sup>Flight and traffic information are obtained through direct feeds updated every few minutes. Other information is obtained from the Web either in real time or through frequent Web harvesting.

(A) AUDIO → RECOGNITION → CONTEXT\_TRACKING\_IN → *TURN\_MANAGEMENT* → CONTEXT\_TRACKING\_OUT → REPLY

(B) RULE: :request\_frame & :eform & :domain Jupiter --> jupiter.turn\_management  
RETRIEVE: :turn\_management\_state  
IN: :request\_frame :eform :parse\_frame :session\_id :filter\_list :nbest\_list  
OUT: :reply\_frame :discourse\_update :system\_initiative :filter\_list :map :list :image :html  
STORE: :turn\_management\_state :list :html :filter\_list :map :image

**Figure 1:** A sequence of operations in a dialogue turn with an example of a rule for the operation “TURN\_MANAGEMENT”.

---

aged 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. The variables are all recorded in a hub-internal *master token*. The conditions consist of simple logical and/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.

While most of the communication among the servers involves an indirect path through the hub, it is possible to have the hub *broker* the relationship between two servers, for example, when the bit rate is very high (e.g., waveforms, images). Thus, an audio server in our system transmits its waveforms directly to a designated recognizer.

Each individual user is associated with a unique *session*; user state information, such as the current language, domain, etc., is recorded via session variables. Each session is usually associated with a particular GUI and/or audio server. Discourse context is organized utterance-by-utterance within a session. Tokens associated with different sessions compete for available resources, and are queued up by the hub when requested servers are busy.

Figure 1 schematizes a typical path through a dialogue turn. Part (A) shows a sequence of operations, each of which is associated with a rule. Part (B) expands the rule for the italicized operation, *TURN\_MANAGEMENT*.

### 2.1. Interfaces between Recognizer and Parser

Our recognizer (SUMMIT, [2]) can produce both an *N*-best list and a word graph as output. *N*-best theories are sent one at a time, and they can be pipelined in the hub script for additional efficiency. In this case, the natural language (NL) component parses the hypotheses as they arrive, and a separate server, called the “gatherer” accumulates the parsed theories and makes a final decision considering parse status (full parse, robust parse, word spot) and linguistic and acoustic scores. If a decision can be made before the full set is analyzed, a message can be communicated to the NL system to stop parsing the remaining hypotheses.

A word graph can generally represent a large *N*-best list in a much more compact space. Since our NL system is able to parse word graphs efficiently, we are exploring this type of interface. It has the additional advantage that it requires a much simpler hub script, eliminating the need for the gatherer and the abort signal.

The NL system (TINA, [3]) sends a *parse frame* to the hub, which is then passed to the discourse component for interpretation in the context of the preceding dialogue. The resulting *request frame* is then passed to the generation system (GENESIS, [4]) to produce two paraphrase strings, a “linguistic” one representing the contents of the frame in English (or some other designated paraphrase language) and the [key: value] pair format, representing a flattened *E-form* (electronic form) structure that is convenient for interpretation for database retrieval and other domain-specific activities in the turn manager.

The parser decides which domain a new query belongs in. For our English-based systems, we are developing separate recognizers and grammars for each of our domains, such that the parser only needs to decide between the target domain and a “local” domain, specializing in meta queries (see Section 2.4). However, we have configured our Mandarin system such that a single recognizer and a single grammar cover all of the domains, with the parser deciding the target domain query by query.

### 2.2. Turn Management

Our systems are generally configured with a suite of turn managers, each one handling a particular restricted domain. Our current configuration runs with four unique turn managers: Jupiter, for weather, Pegasus for flight status, Mercury for flight browsing and reservations, and Voyager for urban navigation and traffic updates.

The turn manager is responsible for deciding how to answer the user’s query. It generally makes use of three distinct meaning representations derived from the user’s query: the original parse frame, the discourse-interpreted request frame, and the E-form. Turn managers often make multiple requests to the database server for pieces of information necessary to properly respond to the query. These requests are implemented through module-to-module subdialogues invoking a *db\_query* program.

Turn managers are controlled at the highest level by a scripting language that is a simplified version of the hub scripting language<sup>3</sup>. It uses an identical protocol for tests on variables, but the separate specifications for IN, OUT, STORE, RETRIEVE, etc., are replaced by a single argument, which is a structure containing all the information of use to the turn manager. This includes critically a *dialogue state* frame which is initialized from the E-form request, and augmented, in the course of processing a turn, by the operations defined in the turn manager’s dialogue control tables. Each operation, upon completion, returns one of three possible *move* states: CONTINUE onto the next rule, STOP processing, and RESTART at the first rule.

In our current configurations, a single database server handles all database needs. Typically, our GENESIS system is invoked to transform an E-form into an SQL query, and the query is then sent to the database server for evaluation. The database server connects to our Oracle relational database to retrieve database tuples, which are then returned to the domain server via a frame representation. The database server may also retrieve information directly from the Web.

Once the turn manager receives a reply from the database, it must prepare a semantic frame representing its reply to the user. If the database fails to retrieve a suitable answer, the turn manager may decide to relax constraints and try again. Alternatively, it may need to make several calls to the database to retrieve different parts of a response,

---

<sup>3</sup>In fact, this dialogue control mechanism was a precursor to the hub scripting language development.

formulating a coherent reply describing the multiple contents retrieved. For example, the Pegasus server must associate flights in the schedule database with their counterparts in the status database representing the status of today's flights, accounting for airline code-sharing rules. The turn manager may also need to prepare a discourse update and/or to take the initiative to request some missing piece of information from the user. In the latter case it may need to output a special system-initiative frame to aid in interpretation of fragments in the subsequent turn.

### 2.3. The Meta-level Turn Manager

In addition to its routing responsibilities, the hub also has an embedded turn manager which handles meta-level queries. These include "scratch that," which decrements the history pointer to the preceding utterance, "clear history," which eliminates all history, "no parse," called when the parser fails to interpret a recognizer query, "call me," to handle requests from the GUI interface for a telephone interaction, and "domain switch," a request to change the high-level topic of the dialogue. For example, a domain-switch routine in the built-in turn manager alters the session domain according to the request, which will then direct the next query to the new domain for recognition and parsing. All of the recognizers and grammars support meta-level queries, which are directed to the local domain upon parse analysis.

## 3. SPECIAL INTERACTIONS

There are a number of unusual circumstances which trigger special events in the hub script. These include communications between the GUI server and the audio server, requests that invoke a switch to a different domain, abort requests that demand termination of any pending computations, and activities that involve retrieval and/or re-entention of information from previous utterances. Some of these are discussed in more detail in this section.

### 3.1. Multimodal interactions

One of the challenges that is well addressed by the GALAXY-II architecture is that of managing multimodal interactions. In a multimodal interaction separate control threads need to manage the various input/output modalities. These threads need to be coordinated and synchronized. In our architecture, the execution model of having a set of active tokens for which rules are fired as their conditions are matched has proven effective in supporting the multiple threads. Different tokens correspond to activity in different threads. Tying all the threads for a given user session together is a session identifier in every token. We next describe the specifics on how we have architected support for two multimodal aspects within our systems: handling a user request for a callback via telephone and association of a user mouse-click event with a spoken utterance.

Our typical system supports (1) an audio-only interaction via telephone, (2) a GUI-only interaction via a client embedded in a Web browser, or (3) a combined audio and GUI interaction. In the combined interaction, the user first interfaces to the system via GUI and then instructs the system to "call me at [phone number]." Upon receiving such a request, the program file directs the audio server to initiate a callback and provides a session ID with the request. When the audio server completes the callback, it sends a new token to the hub. The program file then modifies the session state to indicate that an audio channel is associated with this session. The audio server also requests a session lock, indicating to the hub that any

audio requests for this session must be sent to this particular instantiation of the audio server. (The same hub may be connected to multiple audio server instantiations to support multiple users.)

A mouse-click event is associated with a spoken sentence via a simple mechanism. The hub program sets the clicked item specification within the session's state. As long as the mouse-click was received before the spoken utterance proceeds to the context-tracking input stage, the mouse-click will be handled correctly. The context-tracking server reads the clicked item session state as one of its inputs. Note that presently we do not support time alignment of multiple mouse-clicks to a single utterance. The GALAXY-II hub does not preclude this, but additional timing information will need to be passed from the recognizer to the context-tracking stage.

All of these multimodal interactions are handled in a straightforward manner within the GALAXY-II architecture. The parallel execution, multiple token programming model supports these interactions in a simpler manner than would be possible in a traditional programming language such as C.

### 3.2. Dialogue Context Filtering

Whenever the system displays and/or speaks to the user a list of items it has just retrieved, these items become available for discourse reference. It is highly likely that the user will refer to one of them in a subsequent query. Hence it is advantageous to preferentially select a hypothesis that mentions one of these items. This capability is implemented in the hub script by having the turn manager store the list in the history, with the parser retrieving it in the next turn.

This idea can be extended to include not only *prior* context but also *current* and even *future* context. For instance, if a user asks "What is the status of United flight one oh nine from Dallas to Boston?," prior knowledge of the full set of available United flights between the two specified cities can help in selection among competing recognition hypotheses for the flight number. This is currently implemented by repeating the main program's understanding cycle augmented with the list of candidates retrieved from the database, in the case where the originally selected flight number is inconsistent with the source and destination.

In the event that a user asks for the status of a specific flight without mentioning the source and destination, and that flight is not found in the database, the turn manager engages the user in a subdialogue to obtain the missing information. It retains the recognizer hypothesis until the additional information has been obtained. It then dispatches the hypothesis back through the understanding stage along with the newly retrieved context information.

## 4. EVALUATION

We have long been concerned with developing and maintaining a way of continually evaluating our systems, both holistically and at the component level [5]. The new hub architecture has enabled us to streamline the evaluation process by making it subject to the same hub scripts that control all other system functions. It has also allowed us to augment our suite of evaluation metrics, since an evaluation server can take input from any of the other component servers within the system that we wish to evaluate.

### 4.1. Batchmode Server

We have developed a separate server, named "batchmode," whose purpose is to process user queries through

the system off-line. It operate from a variety of different inputs, including orthographic transcriptions,  $N$ -best lists, word graphs, parse frames, waveform files, and even system log files created from previous live interactions. A hub script can be configured to produce a log file using any of the above inputs, alone or in combination. A batchmode run often includes calls to a special evaluation server, as described below.

Every conversation with our live systems is recorded in a logfile, at a level of detail that is controlled by the hub script. The script supports the specification of any input or output variables to be written to the log, associated with each rule as it fires. A subsequent evaluation script informs the batchmode server which elements from the logfile are of interest in a particular run. For example, in assessing run-time performance, the batchmode server must extract both the selected hypothesis and the transcription of the user's speech from the logfile. The GALAXY-II architecture, combined with the hub scripting language, made control straightforward for this type of logfile evaluation.

We can also use the batchmode server to reprocess stored waveform files. In this case, the batchmode server behaves like an audio server, invoking the module-to-module communication protocol to connect to a recognizer. The recognizer processes the stored waveform file as it would any other utterance, i.e., producing either an  $N$ -best hypothesis or a word graph. This representation is sent back to the hub where it follows the path determined by a hub script for processing.

#### 4.2. Evaluation Server

Assessments of system components are inherently three-way comparisons among the transcription of the spoken utterance and the outputs of old and new versions of the components being evaluated. Our original evaluation protocol functioned at two levels: (1) as a means of benchmarking system performance at the time of data collection and (2) for evaluating performance of new versions of various system components. The first of these is run from a logfile created during data collection. Under the old evaluation paradigm, the second one was also run from a logfile. However, the creation of the logfile involved multiple steps to gather the new data from various system components (e.g., new  $N$ -best hypotheses from the recognizer, new parse frames from the parser) and reformatting the data for the evaluation routine.

Within the new framework, we have developed a separate evaluation server for performing comparisons and accumulating performance statistics. This server can assess both recognition performance (i.e., word accuracy) and understanding, based on the E-form representation. We can easily configure hub scripts that run multiple versions of the same server, to compare new versions of the recognizer against old ones, for example, or to compare two versions of a particular grammar. Furthermore, the rules of a hub script provide a clear tabulation of the parameters being evaluated.

For assessing overall system understanding, we have written a hub script that first uses the batchmode server to process a logfile utterance-by-utterance, sending both a hypothesis and a transcription to the hub, with subsequent routing to TINA and GENESIS. Once the appropriate inputs are created, the hub script sends them to the evaluation server, where they are used to assign scores. The results are returned to the hub script along with all other relevant data for a particular utterance for logging purposes. The evaluation server also outputs cumulative

statistics at the end of each batch run.

Among the evaluation experiments we have run are: (1) logfile recognition/understanding performance at run-time, (2) word graphs compared with  $N$ -best lists, and (3) comparisons of old and new versions of a grammar. We have been able to use the same hub script for evaluating many different types of output, because the hub rules enable us to specify that certain functions be called only when certain variables are present.

## 5. FUTURE WORK

Over the past year, we invested significant resources towards the development of the GALAXY-II architecture, but we feel that the result was well worthwhile. We have generally found that the flexibility inherent in hub scripting empowers us to conceive of systems that we formerly considered to be impossibly complex. We have recently envisioned two extensions to our GALAXY-II system that we feel will be relatively straightforward to implement.

The first would be a wizard-mode system that makes use of two telephone interfaces, one of which records input from the user and speaks system responses to the same user, the other of which plays these same user queries to a wizard who then speaks a "translation" of the user query, either in a simplified form in the same language, or into a different language that the system understand. The wizard's query gets processed through usual channels, except that the audio output is directed to the user, and the GUI output is displayed on the wizard's screen. We believe that such a set-up, which will be a powerful mode for collecting user data for maturing systems, can be configured completely within the hub script.

A second mode that we would like to explore is a system that behaves as an *agent*, calling back a user when a predesignated condition is met. We envision an "agent" server, which might be monitoring a number of events, such as a particular flight or the traffic on a given highway. As soon as the specified condition has occurred (the flight has arrived, the traffic is at a standstill, etc.) the agent server issues a "call me" request, providing both the appropriate phone number and the appropriate "welcome" message, detailing the outcome of the event.

## 6. REFERENCES

- [1] Seneff, S., E. Hurley, R. Lau, C. Pao, P. Schmid, and V. Zue, "GALAXY-II: A Reference Architecture for Conversational System Development," In *Proc. ICSLP '98*, pp. 931-934. Sydney, Australia, 1998.
- [2] J. Glass, J. Chang, and M. McCandless, "A probabilistic framework for feature-based speech recognition," *Proc. ICSLP '96*, Philadelphia, PA, pp. 2277-2280, October, 1996.
- [3] Seneff, S. "TINA: a natural language system for spoken language applications," In *Computational Linguistics*, 18(1), pp. 61-86, 1992.
- [4] Glass, J., J. Polifroni, and S. Seneff, "Multilingual language generation across multiple domains," In *Proc. ICSLP 94*, pp. 983-986, Yokohama, Japan, 1994.
- [5] Polifroni, J., S. Seneff, J. Glass, and T.J. Hazen. 1998. "Evaluation methodology for a telephone-based conversational system." In *Proc. LREC '98*, pp. 43-50, Granada, Spain, 1998.