

# Multi-tasking and Collaborative Activities in Dialogue Systems

Oliver Lemon, Alexander Gruenstein, Alexis Battle, and Stanley Peters

Center for the Study of Language and Information

Stanford University, CA 94306

lemon,alexgru,ajbattle,peters@csl.stanford.edu

## Abstract

We explain dialogue management techniques for collaborative activities with humans, involving multiple concurrent tasks. Conversational context for multiple concurrent activities is represented using a “Dialogue Move Tree” and an “Activity Tree” which support multiple interleaved threads of dialogue about different activities and their execution status. We also describe the incremental message selection, aggregation, and generation method employed in the system.

## 1 Introduction

This paper describes implemented multi-modal dialogue systems<sup>1</sup> which support collaboration with autonomous devices in their execution of multiple concurrent tasks. We will focus on the particular modelling and processing aspects which allow the systems to handle dialogues about multiple concurrent tasks in a coherent and natural manner. Many conversations between humans have this property, and dialogues between humans and semi-autonomous devices will have this feature in as much as devices are able to carry out activities concurrently. This ability to easily interleave communication streams is a very useful property of conversational interactions. Humans are particularly adept

---

<sup>1</sup>This research was (partially) funded under the Wallenberg laboratory for research on Information Technology and Autonomous Systems (WITAS) Project, Linköping University, by the Wallenberg Foundation, Sweden.

at carrying out conversations with multiple threads, or topics, and this capability enables fluid and efficient communication, and thus effective co-ordination of actions. We will show how to endow a dialogue system with some of these capabilities.

The main issues which we address in this paper are:

- Representation of dialogue context such that collaborative activities and multi-tasking are supported.
- Dialogue management methods such that free and natural communication over several conversational topics is supported.
- Natural generation of messages in multi-tasking collaborative dialogues.

In Section 2 we discuss the demands of multi-tasking and collaboration with autonomous devices. Section 3 covers the robot with which our current dialogue system interacts, and the architecture of the dialogue system. In Section 4 we introduce the “joint activities” and Activity Models which represent collaborative tasks and handle multi-tasking in an interface layer between the dialogue system and autonomous devices. Section 5 presents the dialogue modelling and management techniques used to handle multiple topics and collaborative activities. Section 6 surveys the message selection, aggregation, and generation component of the system, in the context of multi-tasking.

## 2 Multi-tasking and Collaboration

A useful dialogue system for interaction with autonomous devices will enable collaboration

with humans in the planning and execution of tasks. Dialogue will be used to specify and clarify instructions and goals for the device, to monitor its progress, and also to jointly solve problems. Before we deal with such issues in detail, we note that such devices also have the following properties which are relevant from the point of view of dialogue management:

- Device sensors may give rise to new information at any time, and this may need to be communicated urgently.
- Devices may perform multiple concurrent activities which may succeed, fail, become cancelled, or be revised. These activities can be topics of conversation.
- Devices exist within dynamic environments, where new objects appear and are available for discussion.

(Allen et al., 2001) present a taxonomy of dialogue systems ranging from “finite-state script” dialogues for simple tasks (such as making a long-distance call) to the most complex “agent-based models” which cover dialogues where different possibilities, such as future plans, are discussed. Within this taxonomy, a useful dialogue system for interaction with autonomous devices must be located at or near the “agent-based” point since we wish to communicate with devices about their possible actions, their plans, and the tasks they are currently attempting. For these reasons we built a dialogue manager that represents (possibly collaborative) activities and their execution status, and tracks multiple threads of dialogue about concurrent and planned activities.

For these sorts of reasons it is clear that form-filling or data-base query style dialogues (e.g. the CSLU Toolkit, (McTear, 1998)) will not suffice here (see (Elio and Haddadi, 1999; Allen et al., 2001) for similar arguments).

### 3 The WITAS Dialogue System

In our current application, the autonomous system is the WITAS<sup>2</sup> UAV (‘unmanned

---

<sup>2</sup>See <http://www.ida.liu.se/ext/witas>

aerial vehicle’) – a small robotic helicopter with on-board planning and deliberative systems, and vision capabilities (for details see e.g. (Doherty et al., 2000)). This robot helicopter will ultimately be controlled by the dialogue system developed at CSLI, though at the moment we interact with a simulated<sup>3</sup> UAV. Mission goals are provided by a human operator, and an on-board planning system then responds. While the helicopter is airborne, an on-board active vision system interprets the scene or focus below to interpret ongoing events, which may be reported (via NL generation) to the operator (see Section 6). The robot can carry out various “activities” such as flying to a location, or following a vehicle, or landing. These activities are specified by the user during dialogue, or can be initiated by the UAV’s on-board AI. In any case, a major component of the dialogue, and a way of maintaining its coherence, is tracking the state of current or planned activities of the device.

A more interesting and problematic notion is that of “joint-activities” between an autonomous system and a human operator. These are activities which the autonomous system cannot complete alone, but which require some human intervention. In our current scenarios, the UAV’s vision system is not good enough to determine whether a particular vehicle is the one sought-after, and only the human operator has the authority to determine this, so that human and robot must collaborate in order to find and track a vehicle. The dialogue in Figure 2 shows how a typical interaction works<sup>4</sup> (other capabilities, such as clarification subdialogues, are covered in (Lemon et al., 2001)). Note here that the user is able to make explicit queries about the robot’s activities (both current and future), that there are concurrent activities, and that conversational initiative centers around the

---

<sup>3</sup>Our UAV simulator uses KIF statements under JTP (the Java Theorem Prover) to represent and non-monotonically update UAV state information.

<sup>4</sup>The system runs on a laptop computer under Windows 2000. Video footage of the system can be found at <http://www-csli.stanford.edu/semlab/witas/>

joint activities currently being specified and executed.

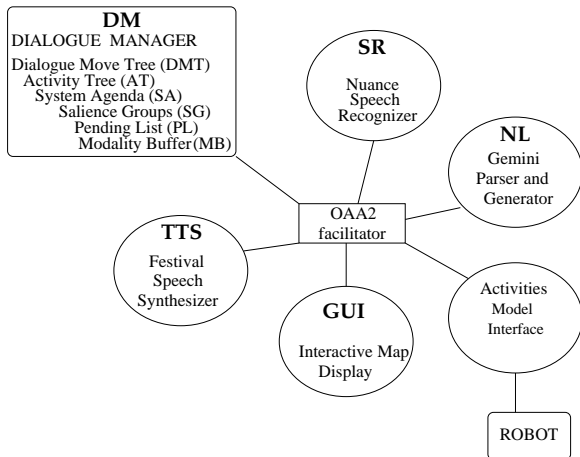


Figure 1: The WITAS dialogue system architecture

## 4 Activity Models

The idea of Activity Modelling in our system is the vision that dialogue systems can, in generality, be built for ‘devices’ which carry out certain well-defined activities (e.g. switch lights on, record on channel  $n$ , send email  $E$  to  $X$ , search for vehicle  $v$ ), and that an important part of the dialogue context to be modelled in such a system is the device’s planned activities, current activities, and their execution status<sup>5</sup>. We choose to focus on building this class of dialogue systems because we share with (Allen et al., 2001), a version of the the *Practical Dialogue Hypothesis*:

“The conversational competence required for practical dialogues, although still complex, is significantly simpler to achieve than general human conversational competence.”

We also share with (Rich et al., 2001) the idea that declarative descriptions of the goal decomposition of activities (COLLAGEN’s “recipes”, our “Activity Models”) are a vital layer of representation, between a dialogue system and the device with which it interacts.

<sup>5</sup>Compare this with the motivation behind the “Pragmatic Adapter” idea of (LuperFoy et al., 1998).

In general we assume that a device is capable of performing some “atomic” activities or actions (possibly simultaneously), which are the lowest-level actions that it can perform. Some devices will only know how to carry out sequences of atomic activities, in which case it is the dialogue system’s job to decompose linguistically specified high-level activities (e.g. “record the film on channel 4 tonight”) into a sequence of appropriate atomic actions for the device. In this case the dialogue system is provided with a declarative “Activities Model” (see e.g. Figure 3) for the device which states how high-level linguistically-specified activities can be decomposed into sequences of atomic actions. This model contains traditional planning constraints such as preconditions and postconditions of actions. In this way, a relatively “stupid” device (i.e. with little or no planning capabilities) can be made into a more intelligent device when it is dialogue-enabled.

At the other end of the spectrum, more intelligent devices are able to plan their own sequences of atomic actions, based on some higher level input. In this case, it is the dialogue system’s role to translate natural language into constraints (including temporal constraints) that the device’s planner recognizes. The device itself then carries out planning, and informs the dialogue manager of the sequence of activities that it proposes. Dialogue can then be used to re-specify constraints, revise activities, and monitor the progress of tasks. We propose that the process of decomposing a linguistically specified command (e.g. “vacuum in the main bedroom and the lounge, and before that, the hall”) into an appropriate sequence of constraints for the device’s on-board planner, is an aspect of “conversational intelligence” that can be added to devices by dialogue-enabling them.

We are developing one representation and reasoning scheme to cover this spectrum of cases from devices with no planning capabilities to some more impressive on-board AI. Both dialogue manager and robot/device have access to a single “Activity Tree” which

Figure 2: A demonstration of the WITAS dialogue system

Multi-modal Utterances	Dialogue Moves
User (U): <i>Our job is to look for a red car</i>	Raise Joint Activity
Robot (R): <i>Ok. I am looking for one.</i>	Confirm Activity
U: <i>Fly here please [+click on map]</i>	Deictic command
R: <i>Okay. I will fly to waypoint one</i>	Confirm Activity, introduce referent
R: <i>Now taking off and flying there.</i>	Report Multi-tasking Activity
U: <i>Stop that. Go to the tower instead.</i>	Command, Revision
R: <i>I have cancelled flying to waypoint one. I will fly to the tower.</i>	Report Activity status.
U: <i>What are you doing?</i>	Activity query
R: <i>I am searching for a red car and flying to the tower</i>	Answer
U: <i>What will you do next?</i>	Activity query
R: <i>I have nothing planned.</i>	Answer
R: <i>I see a red car on main street [display on map, show video images], Is this the right car?</i>	Report, Joint-Activity query
U: <i>Yes, that's the right car</i>	Confirm complete activity
R: <i>Okay. I am following it .</i>	Report initiated activity

is a shared representation of current and planned activities and their execution status, involving temporal and hierarchical ordering (in fact, one can think of the Activity Tree as a Hierarchical Task Network for the device). This tree is built top-down by processing verbal input from the user, and its nodes are then expanded by the device’s planner (if it has one). In cases where no planner exists, the dialogue manager itself expands the whole tree (via the Activity Model for the device) until only leaves with atomic actions are left for the device to execute in sequence. The device reports completion of activities that it is performing and any errors that occur for an activity.

Note that because the device and dialogue system share the same representation of the device’s activities, they are always properly coordinated. They also share responsibility for different aspects of constructing and managing the whole Activity Tree. Note also that some activities can themselves be speech acts, and that this allows us to build collaborative dialogue into the system. For example, in Figure 3 the *ask-complete* activity is a speech act, generating a *yes-no question* to be answered by the user.

#### 4.1 An example Activity Model

An example *locate* activity model for the UAV is shown in Figure 3. It is used when constructing parts of the activity tree involving *locate* commands such as “search for”, “look for” and so on. For instance, if the user says “We’re looking for a truck”, that utterance is parsed into a logical form involving the structure (**locate**, **np[det(a),truck]**).

The dialogue manager then accesses the Activity Model for *locate* and adds a node to the Activity Tree describing it. The Activity Model specifies what sub-activities should be invoked, and under what conditions they should be invoked, what the postconditions of the activity are. Activity Models are similar to the “recipes” of (Rich et al., 2001). For example, in Figure 3 the Activity Model for *locate* states that,

- it uses the *camera* resource (so that any other activity using the camera must be suspended, or a dialogue about resource conflict must be initiated),
- that the preconditions of the activity are that the UAV must be airborne, with fuel and engine indicators satisfactory,
- that the whole activity can be skipped

if the UAV is already “locked-on” to the sought object,

- that the postcondition of the activity is that the UAV is “locked-on” to the sought object,
- that the activity breaks into three sequential sub-activities: watch-for, follow, and ask-complete.

Nodes on the Activity Tree can be either: active, complete, failed, suspended, or canceled. Any change in the state of a node (typically because of a report from the robot) is placed onto the System Agenda (see Section 5) for possible verbal report to the user, via the message selection and generation module (see Section 6).

## 5 The Dialogue Context Model

Dialogue management falls into two parts – dialogue modelling (representation), and dialogue control (algorithm). In this section we focus on the representational aspects, and section 5.2 surveys the main algorithms. As a representation of conversational context, the dialogue manager uses the following data structures which make up the dialogue Information State (*IS*);

- Dialogue Move Tree (DMT)
- Activity Tree (AT)
- System Agenda (SA)
- Pending List (PL)
- Salience Groups (SG)
- Modality Buffer (MB)

Figure 4 shows how the Dialogue Move Tree relates to other parts of the dialogue manager as a whole. The solid arrows represent possible update functions, and the dashed arrows represent query functions. For example, the Dialogue Move Tree can update Salience Groups, System Agenda, Pending List, and Activity Tree, while the Activity Tree can update only the System Agenda and

send execution requests to the robot, and it can query the Activity Model (when adding nodes). Likewise, the Message Generation component queries the System Agenda and the Pending List, and updates the Dialogue Move Tree whenever a synthesized utterance is produced.

Figure 5 shows an example Information State logged by the system, displaying the interpretation of the system’s utterance “now taking off” as a report about an ongoing “go to the tower” activity (the Pending List and System Agenda are empty, and thus are not shown).

### 5.1 The Dialogue Move Tree

Dialogue management uses a set of abstract dialogue move classes which are domain independent (e.g. *command*, *activity-query*, *wh-question*, *revision*, ...). Any ongoing dialogue constructs a particular Dialogue Move Tree (DMT) representing the current state of the conversation, whose nodes are instances of the dialogue move classes, and which are linked to nodes on the Activity Tree where appropriate, via an *activity tag* (see below).

Incoming logical forms (LFs) from the parsing process are always tagged with a dialogue move (see e.g. (Ginzburg et al., 2001)), which precedes more detailed information about an utterance. For instance the logical form: **command**([go], [param-list ([pp-loc(to), arg([np(det([def],the), [n(tower,sg)]))])])])

corresponds to the utterance “go to the tower”, which is flagged as a *command*.

A slightly more complex example is; **report**(inform, agent([np([n(uav,sg)])]), compl-activity([command([take-off])])

which corresponds to “I have taken off” – a report from the UAV about a completed ‘taking-off’ activity.

The first problem in dialogue management is to figure out how these incoming “Conversational Moves” relate to the current dialogue context. In other words, what dialogue moves do they constitute, and how do they relate to previous moves in the conversation? In particular, given multi-tasking, to which thread

Figure 3: A “Locate” Activity Model for a UAV, exhibiting collaborative dialogue

```

task Locate // locate is "find-by-type", collaborative activity.
// Breaks find into subactivities: watch_for, follow, ask_complete.
{ResourcesUsed {camera;} // will be checked for conflicts.
PreConditions //check truth of KIF statements.
  {(Status flight inair) (Status engine ok) (Status fuel ok);}
SkipConditions // skip this Activity if KIF condition true.
  {(Status locked-on THIS.np);}
PostConditions// assert these KIF statements when completed.
  {(Status locked-on THIS.np) };
Children SEQ //sequential sub-activities.
  {TaskProperties
    {command = "watch_for"; // basic robot action ---
      np = THIS.np;} // set sensors to search.
  TaskProperties
    {command = "follow_phobj";//triggers another complex activity
      np = THIS.np;} // following a candidate object.
  TaskProperties // collaborative dialogue action:
    {command = "ask_complete";// asks user whether this is
      np = THIS.np; }}} // the object we are looking for.

```

of the conversation does an incoming utterance belong? We use the Dialogue Move Tree to answer these questions:

1. A DMT is a history or “message board” of dialogue contributions, organized by “thread”, based on activities.
2. A DMT classifies *which* incoming utterances can be interpreted in the current dialogue context, and which cannot be. It thus delimits a space of possible Information State update functions.
3. A DMT has an *Active Node List* which controls the order in which this function space is searched <sup>6</sup>.
4. A DMT classifies *how* incoming utterances are to be interpreted in the current dialogue context.

In general, then, we can think of the DMT as representing a function space of dialogue

<sup>6</sup>It also defines an ordering on language models for speech recognition.

Information State update functions. The details of any particular update function are determined by the node type (e.g. *command*, *question*) and incoming dialogue move type and their contents, as well as the values of *Activity Tag* and *Agent*. We discuss this further below.

## 5.2 Interpretation and State Update

The central algorithm controlling dialogue management has two main steps, *Attachment*, and *Process Node*;

1. *Attachment*: Process incoming input conversational move *c* with respect to the current DMT and Active Node List, and “attach” a new node *N* interpreting *c* to the tree if possible.
2. *Process Node*: process the new node *N*, if it exists, with respect to the current information state. Perform an Information State update using the dialogue move type and content of *N*.

It is worth noting that the node type created after attachment may not be the same as

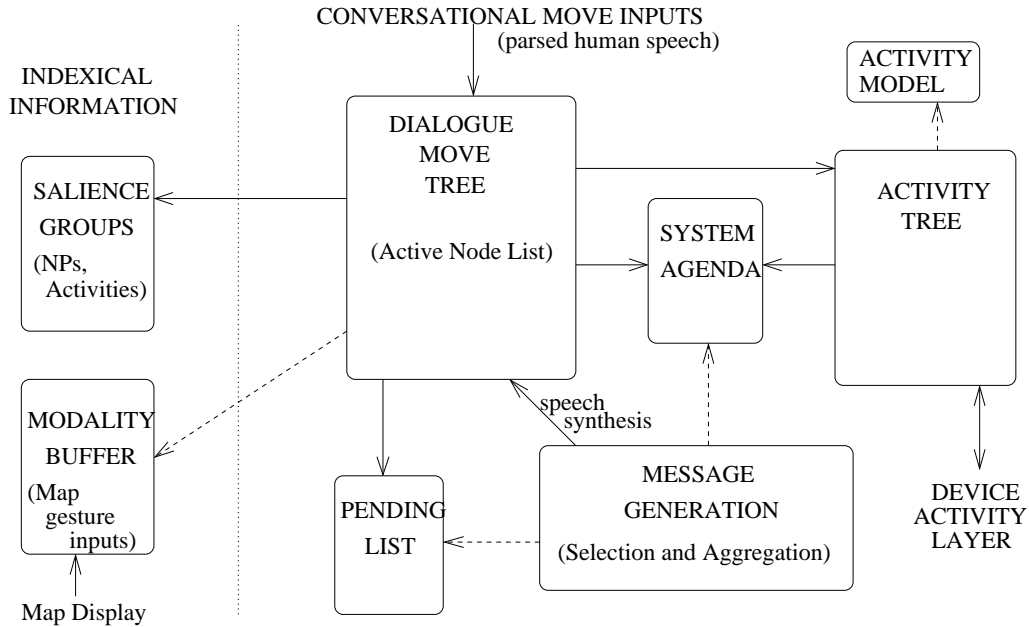


Figure 4: Dialogue Manager Architecture (solid arrows denote possible updates, dashed arrows represent possible queries)

the dialogue move type of the incoming conversational move  $c$ . Depending on the particular node which attaches the new input, and the move type of that input, the created node may be of a different type. For example, if a *wh-question* node attaches an input which is simply a *command*, the *wh-question* node may interpret the input as an answer, and attach a *wh-answer*. These interpretation rules are local to the node to which the input is attached. In this way, the DMT interprets new input in context, and the pragmatics of each new input is contextually determined, rather than completely specified via parsing using conversational move types.

When an update function  $g$  exists, its effects depend on the details of the incoming input  $c$  (in particular, to the dialogue move type and the contents of the logical form) and the DMT node to which it attaches. The possible attachments can be thought of as adjacency pairs, and each dialogue move class contains information about which node types it can attach. For instance the *command* node type can attach *confirmation*, *yn-question*, *wh-question*, and *report* nodes. Examples of different attachments available in our current

system can be seen in Figure 6. For example, the rows for *wh-question* nodes state that:

- a *wh-question* by the system with activity tag  $t$  can attach a user’s *wh-answer* (if it is a possible answer for that activity)
- a user’s *wh-question* with (possibly empty) activity tag  $t$  can attach a system *wh-answer* for that activity.

Note that Figure 6 does *not* state what move type new input is attached as, when it is attached (as we mentioned earlier, attachment can result in a change of move type for the new input).

These possible attachments delimit the ways in which dialogue move trees can grow, and thus classify the dialogue structures which can be captured in the current system. As new dialogue move types are added to the system, this table is being extended to cover other conversation types (e.g. tutoring (Clark et al., 2001)).

## 6 Message generation

Since the robot is potentially carrying out multiple activities at once, a particular prob-

lem is how to determine appropriate generation of utterances about those activities, in a way which does not overload the user with information, yet which establishes and maintains appropriate context in a natural way.

Generation for dialogue systems in general is problematic in that dialogue contributions arise incrementally, often in response to another participant's utterances. For this reason, generation of large pieces of text is not appropriate, especially since the user is able to interrupt the system. Other differences abound, for example that aggregation rules must be sensitive to incremental aspects of message generation.

As well as the general problems of message selection and aggregation in dialogue systems, this particular type of application domain presents specific problems in comparison with, say, travel-planning dialogue systems – e.g. (Seneff et al., 1991). An autonomous device will, in general, need to communicate about,

- its perceptions of a changing environment,
- progress towards user-specified goals,
- execution status of activities or tasks,
- its own internal state changes,
- the progress of the dialogue itself.

For these reasons, the message selection and generation component of such a system needs to be of wider coverage and more flexible than template-based approaches, while remaining in real, or near-real, time (Stent, 1999). As well as this, the system must potentially be able to deal with a large bandwidth stream of communications from the robot, and so must be able to intelligently filter them for “relevance” so that the user is not overloaded with unimportant information, or repetitious utterances. In general, the system should appear as ‘natural’ as possible from the user’s point of view – using the same language as the user if possible (“echoing”), using anaphoric referring expressions where possible, and aggregating utterances where appropriate. A

‘natural’ system should also exhibit “variability” in that it can convey the same content in a variety of ways. A further desirable feature is that the system’s generated utterances should be in the coverage of the dialogue system’s speech recognizer, so that system-generated utterances effectively prime the user to speak in-grammar.

Consequently we attempted to implement the following features in message selection and generation: *relevance filtering; recency filtering; echoing; variability; aggregation; symmetry; real-time generation.*

Our general method is to take as inputs to the process various communicative goals of the system, expressed as logical forms, and use them to construct a single new logical form to be input to Gemini’s Semantic Head-Driven Generation algorithm (Shieber et al., 1990), which produces strings for Festival speech synthesis. We now describe how to use complex dialogue context to produce natural generation in multitasking contexts.

## 6.1 Message selection - filtering

Inputs to the selection and generation module are “concept” logical forms describing the communicative goals of the system. These are structures consisting of *context tags* (e.g. activity identifier, dialogue move tree node, turn tag) and a *content logical form* consisting of a Dialogue Move (e.g. report, wh-question), a priority tag (e.g. *warn* or *inform*), and some additional content tags (e.g. for objects referred to). An example input logical form (LF) is, “**report(inform, agent(AgentID), cancel-activity(ActivityID))**”, which corresponds to the report “I have cancelled flying to the tower” when AgentID refers to the robot and ActivityID refers to a “fly to the tower” task.

Items which the system will consider for generation are placed (either directly by the robot, or indirectly by the Activity Tree) on the “System Agenda” (SA), which is the part of the dialogue information state which stores communicative goals of the system. Communicative goals may also exist on the “Pending List” (PL) which is the part of the infor-

mation state which stores questions that the system has asked, but which the user has not answered, so that they may be re-raised by the system. Only questions previously asked by the system can exist on the Pending List.

Due to multi-tasking, at any time there is a number of “Current Activities” which the user and system are performing (e.g. fly to the tower, search for a red car). These activities are topics of conversation (defining threads of the DMT) represented in the dialogue information state, and the system’s reports can be generated by them (in which case they are tagged with that activity label) or can be relevant to an activity in virtue of being about an object which is in focus because it is involved in that activity.

Some system reports are more urgent than others (e.g. “I am running out of fuel”) and these carry the label *warning*. Warnings are always relevant, no matter what activities are current – they always pass the recency and relevance filters.

Echoing (for noun-phrases) is achieved by accessing the Saliency List whenever generating referential terms, and using whatever noun-phrase (if any) the user has previously employed to refer to the object in question. If the object is top of the saliency list, the generator will select an anaphoric expression.

The end result of our selection and aggregation module (see section 6.2) is a fully specified logical form which is to be sent to the Semantic-Head-Driven Generation component of Gemini (Shieber et al., 1990). The bi-directionality of Gemini (i.e. that we use the same grammar for both parsing and generation) automatically confers a useful “symmetry” property on the system – that it only utters sentences which it can also understand. This means that the user will not be misled by the system into employing out-of-vocabulary items, or out-of-grammar constructions. Another side effect of this is that the system utterances prime the user to make in-grammar utterances, thus enhancing co-ordination between user and system in the dialogues.

## 6.2 Incremental aggregation

Aggregation combines and compresses utterances to make them more concise, avoid repetitive language structure, and make the system’s speech more natural and understandable. In a dialogue system aggregation should function incrementally because utterances are generated on the fly. In dialogue systems, when constructing an utterance we often have no information about the utterances that will follow it, and thus the best we can do is to compress it or “retro-aggregate” it with utterances that preceded it. Only occasionally does the System Agenda contain enough unsaid utterances to perform reasonable “pre” aggregation.

Each dialogue move type contains rules for these types of aggregation, described in full detail in (Lemon and Battle, 2002). Each dialogue move type (e.g. report, wh-question) has its own aggregation rules, stored in the class for that LF type. In each type, rules specify which other dialogue move types can aggregate with it, and exactly how aggregation works. The rules note identical portions of LFs and unify them, and then combine the non-identical portions appropriately.

For example, the LF that represents the phrase “I will fly to the tower and I will land at the parking lot”, will be converted to “I will fly to the tower and land at the parking lot” according to the compression rules. Similarly, “I will fly to the tower and fly to the hospital” gets converted to “I will fly to the tower and the hospital”.

## 7 Summary

We explained the dialogue modelling techniques which we implemented in order to build a real-time multi-modal conversational interface to an autonomous device. The novel issues tackled by the system and its dialogue model are that it is able to manage conversations about multiple tasks and collaborative activities in a robust and natural way.

We argued that in the case of dialogues with devices, a dialogue management mechanism has to be particularly robust and flexi-

ble, especially in comparison with finite-state or frame-based dialogue managers which have been developed for information-seeking dialogues, such as travel planning, where topics of conversation are predetermined. Another challenge was that conversations may have multiple open topics at any one time, and this complicates utterance interpretation and generation.

We discussed the dialogue context model and algorithms used to produce a system with the following features:

- supports multi-tasking, multiple topics, and collaboration,
- support of commands, questions, revisions, and reports, over a dynamic environment,
- multi-modal, mixed-initiative, open-ended dialogues,
- echoic and variable message generation, filtered for relevance and recency
- asynchronous, real-time operation.

An video demonstration of the system is available at [www-csli.stanford.edu/semlab/witas/](http://www-csli.stanford.edu/semlab/witas/). A full demonstration will be presented at the conference.

## References

- James Allen, Donna Byron, Myroslva Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. 2001. Toward conversational human-computer interaction. *AI Magazine*, 22(4):27–37.
- Brady Clark, John Fry, Matt Ginzton, Stanley Peters, Heather Pon-Barry, and Zachary Thomsen-Gray. 2001. Automated tutoring dialogues for training in shipboard damage control. In *Proceedings of SIGDIAL 2001*.
- Patrick Doherty, Gösta Granlund, Krzysztof Kuchcinski, Erik Sandewall, Klas Nordberg, Erik Skarman, and Johan Wiklund. 2000. The WITAS unmanned aerial vehicle project. In *European Conference on Artificial Intelligence (ECAI 2000)*.
- Renee Elio and Afsaneh Haddadi. 1999. On abstract task models and conversation policies. In *Workshop on Specifying and Implementing Conversation Policies, Autonomous Agents'99*, Seattle.
- Jonathan Ginzburg, Ivan A. Sag, and Matthew Purver. 2001. Integrating Conversational Move Types in the Grammar of Conversation. In *BI-DIALOG 2001—Proceedings of the 5th Workshop on Formal Semantics and Pragmatics of Dialogue*, pages 45–56.
- Oliver Lemon and Alexis Battle. 2002. Message selection, generation, and aggregation in a multi-modal dialogue system. In *preparation*.
- Oliver Lemon, Anne Bracy, Alexander Gruenstein, and Stanley Peters. 2001. Information states in a multi-modal dialogue system for human-robot conversation. In Peter Kühnlein, Hans Reiser, and Henk Zeevat, editors, *5th Workshop on Formal Semantics and Pragmatics of Dialogue (Bi-Dialog 2001)*, pages 57 – 67.
- Susann LuperFoy, Dan Loehr, David Duff, Keith Miller, Florence Reeder, and Lisa Harper. 1998. An architecture for dialogue management, context tracking, and pragmatic adaptation in spoken dialogue systems. In *COLING-ACL*, pages 794 – 801.
- M. McTear. 1998. Modelling spoken dialogues with state transition diagrams: Experiences with the CSLU toolkit. In *Proc 5th International Conference on Spoken Language Processing*.
- Charles Rich, Candace Sidner, and Neal Lesh. 2001. Collagen: applying collaborative discourse theory to human-computer interaction. *AI Magazine*, 22(4):15–25.
- S. Seneff, L. Hirschman, and V. W. Zue. 1991. Interactive problem solving and dialogue in the ATIS domain. In *Proceedings of the Fourth DARPA Speech and Natural Language Workshop*. Morgan Kaufmann.
- Stuart M. Shieber, Gertjan van Noord, Fernando C. N. Pereira, and Robert C. Moore. 1990. Semantic-head-driven generation. *Computational Linguistics*, 16(1):30–42.
- Amanda Stent. 1999. Content planning and generation in continuous-speech spoken dialog systems. In *Proceedings of KI'99 workshop "May I Speak Freely?"*.

Figure 5: A snapshot of an Information State (from the HTML system logs)

```

Utterance: ‘‘now taking off’’ (by System 11/7/01 4:50 PM)
Conversational Move:
report(inform,agent([np([n(uav,sg)])]),curr_activity([command([take_off])]))

Dialogue Move Tree (position on active node list in parens [0 = most active])
* Root (1)
  Root
  o Command (0)
    command([go],[param_list([pp_loc(to,arg([np(det([def],the),[n(tower,sg)]
    )])))])) [[dmtask0] current]
    + Report
      report(inform,agent([np([n(uav,sg)])]),curr_activity([command
      ([take_off])])) []
  o Report
    report(inform,agent([np([n(uav,sg)])]),confirm_activity([command([go],
    [param_list([pp_loc(to,arg([np(det([def],the),[n(tower,sg)],
    )])))])))])) [[dmtask0] current]

Activity Tree
* root
  o [dmtask0] current
    relation = SEquential
    command = go
    pp = pp_loc(to,Args)
    np = np(det([def],the),[n(tower,sg)])
    + [sim3] current
      relation = none
      command = take_off
      pp = null, np = null

Salience List (least salient -- most salient)
* [np(det([def],the),[n(tower,sg)])] (speech)
* [np(det([def],the),[n(tower,sg)])] (speech)

Salient Activities (least salient -- most salient)
* [dmtask0] current
* [sim3] current

```

Figure 6: Attachment in the Dialogue Move Classes

Node Type	Activity Tag	Speaker	Attaches
command	t	user	confirmation(t), y-n question(t), wh-question(t), report(t),
confirmation	t	system	command(t)
report	t	system	wh-answer(t, user),
wh-question	t	system	wh-answer(t, system)
wh-question	t	user	yn-answer(t, user),
yn-question	t	system	wh-question(t, system)
revision	t	user	confirmation(t,system)
yn-answer	t	user	confirmation
wh-answer	any	user/system	confirmation
root	null	null	command, question, report, revision

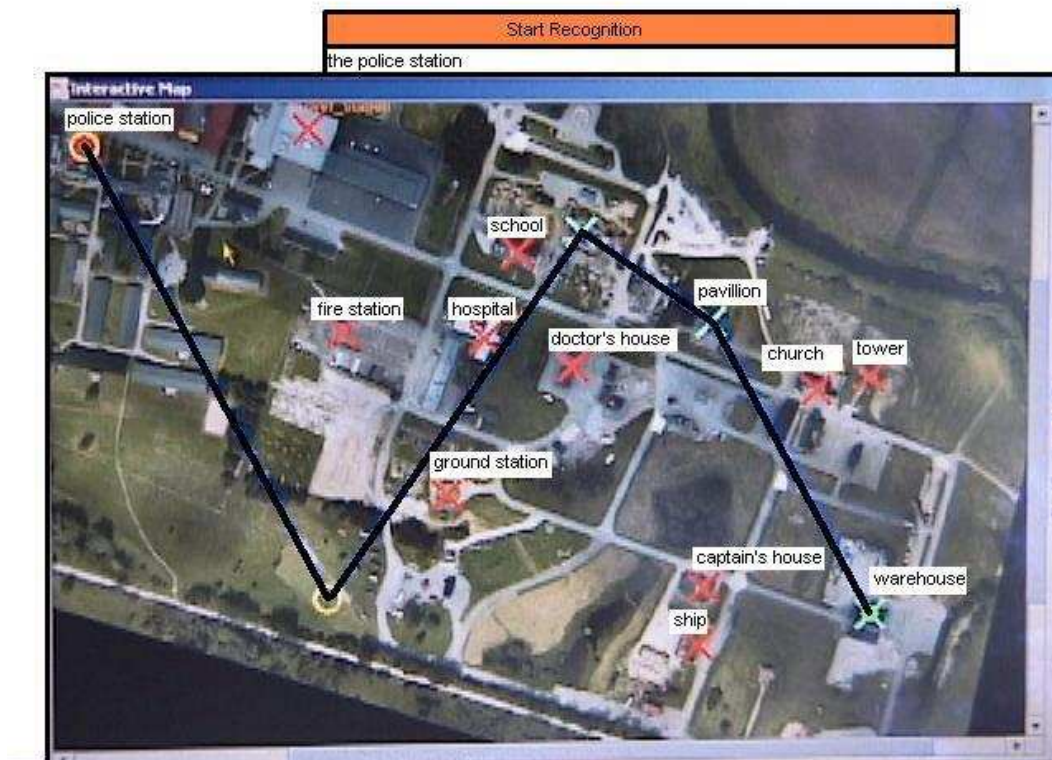


Figure 7: Part of the Graphical User Interface, showing a flight plan