Planning in the JAVELIN QA System

Laurie S. Hiyakumoto
May 2004
CMU-CS-04-132

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

The Planner Module of the JAVELIN Question-Answering system is responsible for sequencing actions in the question-answering process and controlling their execution. This document describes the current implementation of the Planner Module based on the PLEXIS (**Pl**anning and **Ex**ecution for **I**nformation **S**paces) planner, the protocols it uses to communicate with the rest of JAVELIN, and the model of the question-answering process on which planning and execution is based. Instructions for installing and testing the server are provided, along with a brief discussion of future research directions.

This research has been conducted under the supervision of Jaime G. Carbonell and Manuela M. Veloso. This research was supported in part by the Advanced Research and Development Activity (ARDA)'s Advanced Question Answering for Intelligence (AQUAINT) Program under contract MDA908-02-C-0009, and by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL) under Grant No. F30602-00-2-0549. The content of this publication does not necessarily reflect the position of the funding agencies and no official endorsement should be inferred.

Keywords: planning, execution, question-answering, PLEXIS , JAVELIN

Contents

1 Introduction

The goal of a question-answering (QA) system is to provide a user with an appropriate and succinct answer, given an information request expressed in unrestricted natural language. Systems to date have been remarkably successful at answering trivia-type questions (e.g., *"Where is Belize located?"*) employing ad-hoc combinations of NLP techniques in fixed pipeline architectures. However, such fixed-strategy approaches are likely to fall short when presented with complex requests involving sequences of related questions and user-interaction. Indeed, recent work has demonstrated that even for trivia questions, the use of feedback loops [3] and the incorporation of multiple QA strategies [1, 2] can improve performance. Moreover, it is desirable that future QA systems provide the flexibility to incorporate new NLP tools and knowledge resources as they become available, dynamically selecting the subset whose performance characteristics best match the current request.

The JAVELIN (Justification-based Answer Valuation through Language Interpretation) system aims to address these issues by combining a utility-based planner with a modular, object-oriented QA architecture [4]. As depicted in Figure 1), JAVELIN is implemented as a set of modular QA components, each performing one of the four major QA processing stages distinguished by the system: question analysis, document retrieval, answer candidate extraction, and selection of a final answer. The planner serves as the overall controller, selecting and invoking QA components to maximize the expected utility of the information produced.

This document describes our initial implementation of JAVELIN's planning component, based on the PLEXIS (**Pl**anning and **Ex**ecution for **I**nformation **S**paces) planner. Its focus is the integration of the planner with the rest of the JAVELIN system: the communication protocols the planner uses, the model of the question-answering process on which planning and execution is currently based, and instructions for installing and testing the server. Although a brief overview of the planning and execution algorithm is provided, this document is not intended as a user-guide or tutorial for PLEXIS ; we expect to address that aspect of our work in a future publication.

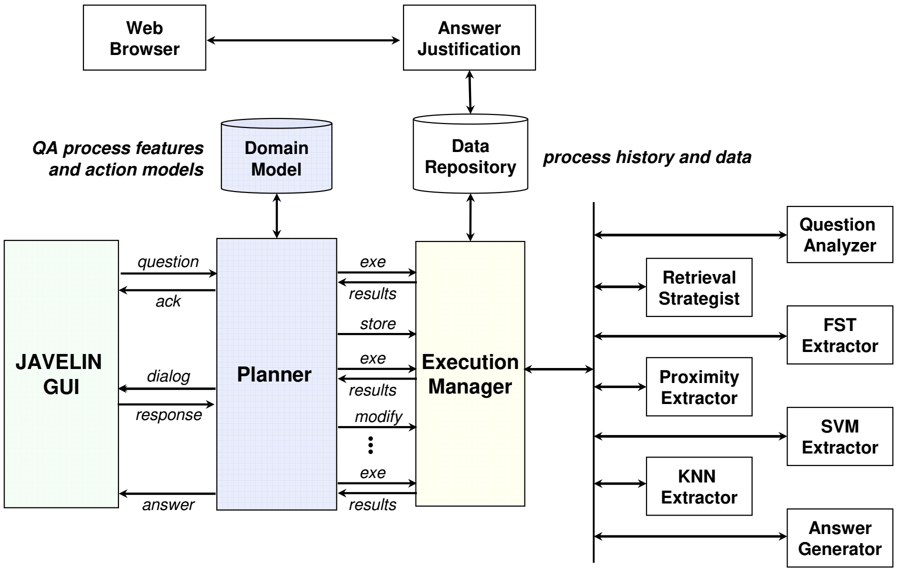

Figure 1: The JAVELIN QA architecture.

2 Planner Module Design

The Planner Module operates as a service for the JAVELIN GUI, and communicates with the rest of the system via the Execution Manager (EM). Internally, it is comprised of two components (Figure 2): the server interface and the PLEXIS forward-chaining state-space planner. The server implements the QA domain-specific functionality, handling communication with the GUI and EM and translating information produced by the QA system into the planner's internal state representation. PLEXIS provides all of the domain-independent planning and execution functionality. Shared between the two are a domain model of the QA process, a problem model of the current question, the server's interface to the EM, which is called by the planner when it needs to execute a step in the plan or save data to the system repository, and an object database for the current planning session, which the planner uses to look up attributes of the information state.

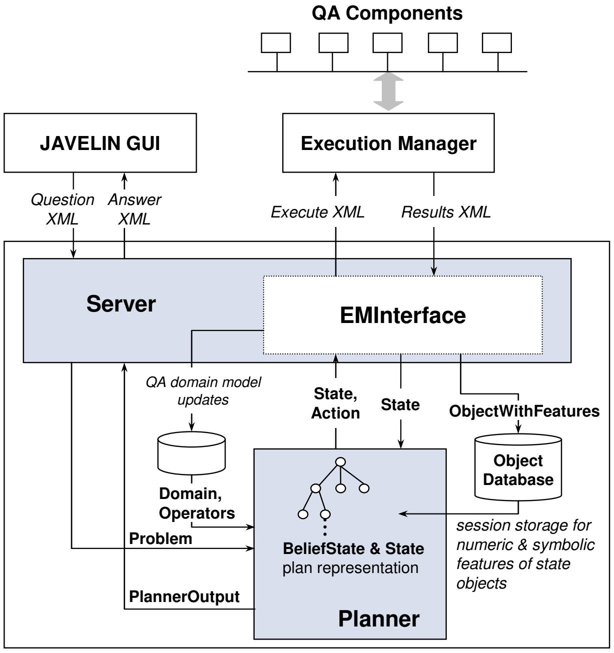

Figure 2: Component and data relationships within the Planner Module. The server component provides the QA domain-specific functionality, and the PLEXIS planner provides domain-independent planning functionality. Major data classes are identified by boldface sans-serif labels. For clarity, user-interaction dialogs and EM data storage requests have been omitted.

2.1 Typical System Interaction

Figure 3 presents an event sequence diagram for a typical question-answering interaction between the planner, the GUI, and the EM (for clarity, data storage requests between the Planner Module and EM are omitted). Upon receiving a new question from the GUI, the Planner Module instructs the EM to call a question analysis component, and uses the resulting analysis to generate a planning problem describing the initial state and information goal. Internally, the Planner Module then calls the PLEXIS planner to begin the planning and execution process, which continues until the goal criteria is met (an answer or set of answers with sufficiently high expected utility is found), or available resources are exhausted. It then returns the answer or a failure message to the GUI.

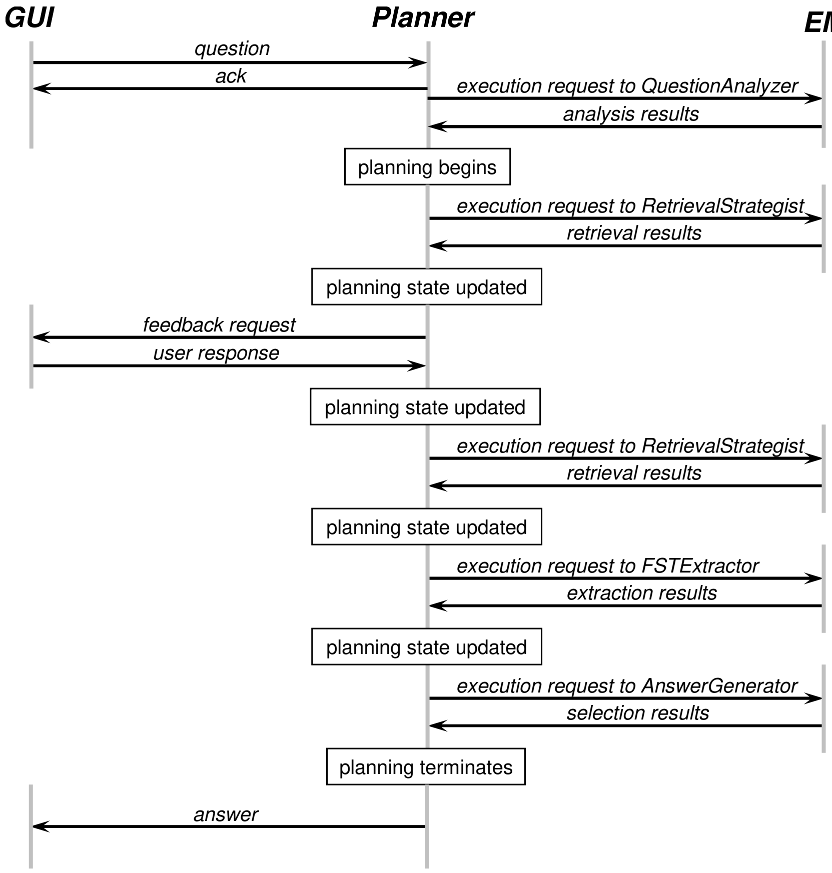

Figure 3: A typical question-answering session event diagram. For clarity, EM data storage requests have been omitted.

Each call made by the Planner Module to the EM represents the execution of a single action in its plan. The planner specifies which QA component the EM should call, and what data should be used to construct the component's input. Results are passed back to the Planner Module, and used to revise the internal model of the information state on which PLEXIS bases subsequent planning decisions.

When PLEXIS is supplied with a model of the QA process that includes user-feedback amongst its possible actions, it can also choose to gather more information from the user using simple dialogs. A detailed description of all communication protocols and data passed between the modules is provided in Section 4.

2.2 The PLEXIS Planning Algorithm

The PLEXIS planner is based on an integrated planning and execution algorithm that performs a best-first search across *belief states*, the set of possible states representing the information content the system may currently possess. The algorithm, presented in Table 1, is supplied with a domain model $\mathcal{D}$ of the QA process, a problem statement defining an initial belief state I, a utility function U, an information goal condition G, a utility threshold γ, a satisfiability threshold σ, and an applicability threshold α. The domain model defines the set of data features that represent information states and the set of atomic actions, or *operators*, the planner has control over. The initial state defines the information (e.g., question attributes) that is known at the start of the planning session. Collectively, the utility function and thresholds determine how the planner searches the space and under what conditions it terminates.

```
PLANANDEXECUTE(D,I,U,G,γ,σ,α)

b ⇐ I
C ⇐ SUCCESSORS(b,D,U,α)
p ⇐ UNEXECUTEDACTIONS(b)

while TRUE
    while PROBSATISFIES(b,U,G,γ,σ) and ¬ EMPTY(p)
        b′⇐ EXECUTE(POPFRONT(p))
        if REPLAN(b′,b,C))
            PLANANDEXECUTE(D,b′,U,G,γ,σ,α)
        else
            (b,C) ⇐ UPDATE(b′,b,C)

    if PROBSATISFIES(b,U,G,γ,σ)
        return success
    else if (EMPTY(C) and EMPTY(p)) or LIMIT()
        return failure
    else if (CHOOSEEXTEND(b,C,D,U,α))
        b ⇐ CHOOSEBESTONE(C)
        C ⇐ (C− b) ∪ SUCCESSORS(b,D,U,α)
    else
        b′⇐ EXECUTE(POPFRONT(p))
        if REPLAN(b′,b,C))
            PLANANDEXECUTE(D,b′,U,G,γ,σ,α)
        else
            (b,C) ⇐ UPDATE(b′,b,C)
```

Table 1: The PLEXIS planning and execution algorithm.

Beginning with the initial belief state as the root, the algorithm evaluates all successor belief states reachable from the current state via a single action and selects the one with the highest *expected utility*, which is equal to the weighted sum of the estimated utilities of each state comprising the belief state:

$$\mathbf{EU}(\mathbf{b}) = \sum_{s \in \text{STATES}(\mathbf{b})} P(s)U(s)$$

The current belief state is updated to reflect the belief state projected by the SUCCESSORS function, and the selection process repeats. The actual *plan*, the sequence of actions that transforms the initial state into the current belief state, is implicitly maintained within each belief state by storing its generating operator and parent belief state.

A pictorial view of what happens during a single step of the belief state projection process is presented in Figure 4. The SUCCESSORS function simulates every possible effect $e_{j_{a_i}}$ that operator a_i may have when applied to every state within the current belief state $\mathbf{b}$. The resulting belief state $\mathbf{b}'$ consists of all possible outcome states (possibly contradictory) and their associated likelihoods.

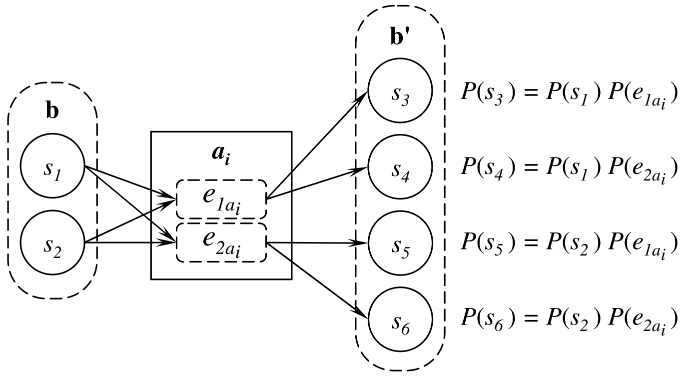

Figure 4: A pictorial view of the belief state projection process.

2.3 Execution

At each step, the PLEXIS algorithm considers the trade-off between executing the first unexecuted action in the plan and continuing to plan with the uncertain outcomes of the projected belief states. If execution is chosen, it is followed by an update of the information state and an assessment of the need for replanning.

The main advantage of interleaving execution is that it can provide additional information about how well the information-gathering process is going, reducing the uncertainty and the number of possible states the planner must consider during forward projection. It may also allow the planner to terminate earlier if it discovers the true information state was much better than its original projections. The main disadvantage of early execution is that resources (e.g., time) cannot be recovered once they are consumed, potentially leading to a less useful result than if planning had continued. In the worst case, where resources are severely limited, executing steps without sufficient look-ahead could result in failure to find *any* solution because the resources are no longer available to complete the task.

Currently, the Planner Module runs PLEXIS with a *reactive* execution strategy: each step is executed immediately after its addition to the plan. Our rationale for making this simplification was twofold. First, in this initial development phase we have been concerned primarily with improving the quality of answers produced rather than the cost of producing them. Executing after each step provides the planner with the maximum amount of information possible during the decision-making process. Second, executing after each step eliminated the need for replanning decisions, enabling us to defer development of a suitable model to a later date.

2.4 Termination

Planning terminates when one of the following three conditions is met: all steps in the plan have been executed and the resulting information state meets the goal satisfaction criteria; the process hits a predefined search limit (e.g., a time limit); or there are no additional planning or execution actions PLEXIS can perform. Note that satisfying the goal criteria does not guarantee the system has produced a correct answer. It only means that the model in use by the Planner Module predicts the answer is correct. Figure 5 presents two plans produced and executed by the Planner Module illustrating this point and demonstrating the planner's ability to produce different plans.

Q: *What movie won the Academy Award for best picture in 1989?*
A: Driving Miss Daisy (correct)

```
<retrieve_documents DS6024 RO6637>
<extract_SVM_candidate_fills FS18637 RO6637 DS6024>
<rank_candidates AL5184 RO6637 FS18637>
<check_answers A5046 AL5184 Q2694>
```

Q: *In which country is Timbuktu?*
A: Japan (wrong)

```
<retrieve_documents DS6265 RO6880>
<extract_FST_candidate_fills FS21076 RO6880 DS6265>
<extract_SVM_candidate_fills FS21080 RO6880 DS6265>
<rank_candidates AL5420 RO6880 FS21080>
<extract_Light_candidate_fills FS21085 RO6880 DS6265>
<rank_candidates AL5421 RO6880 FS21085>
<extract_KNN_candidate_fills FS21087 RO6880 DS6265>
<rank_candidates AL5422 RO6880 FS21087>
<check_answers A5268 AL5422 Q3618>
```

Figure 5: Action sequences generated and executed by the JAVELIN Planner Module.

3 Modeling the QA Process as a Planning Domain

This section describes the components of the current JAVELIN QA planning domain model: the types of manipulable objects and resources defined, the relationships and features of objects used to model the possible information states, and the actions representing calls to the individual components of the QA system. Collectively, it defines a world model for the task of interest (i.e., a model of the question-answering process), capturing characteristics common to all problem-solving sessions within the domain. A copy of a sample domain file for the JAVELIN QA system is provided in Appendix A.

3.1 Types

Types define the categories of objects created and manipulated by the QA process, plus four types automatically defined by PLEXIS for all domains: a generic toptype that serves as the root for all closed-class sets of objects, and a separate three-class hierarchy for numeric values consisting of fluent, int, and float. Each type may have one or more *subtypes*, defining category specializations. These parent-child relationships are equivalent to *ISA* relations. For example, any organization-name *ISA* proper-name.

Figure 6 presents a partial listing of the type hierarchy implemented for the JAVELIN QA domain. In addition to defining the major data objects produced by the system, it also incorporates the system-wide question type (as subtypes of qtype) and answer type hierarchies (as subtypes of atype).

3.2 Constants and Objects

Every information state of a planning problem may contain specific *objects*, instances of a particular type, identified by a unique id. They define the session-specific data used as input to or created as the product a particular action. For example, a specific document set instance may be denoted as docset DS123. *Constants* denote special, persistent objects in the planning domain. They are present in all information states of the planning session, and although they can influence the decisions made, they are neither created nor destroyed by the actions taken.

3.3 Predicates and Features

Predicates define data relationships (e.g., denoting which document set produced a particular answer candidate) and features of the question context and process that we wish to track (e.g., whether or not a session is interactive, and the network availability of a particular module). Each predicate is defined in terms

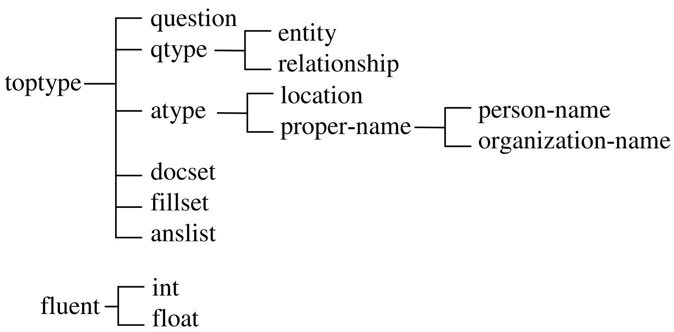

Figure 6: Partial type hierarchy for JAVELIN. The three numeric types (fluent, int, float) and toptype are automatically defined by the PLEXIS planner.

of the relation it represents, and a list of typed arguments (possibly empty), declaring the classes of objects that may possess the relationship. For example, the predicate `(candidate_fills ?fillset ?qtype ?docset ?ix-name)` is used to track which document set (`?docset`), question analysis object (`?qtype`) and extractor version (`?qtype`) were used to create a particular set of answer candidates (`?fillset`).

Features are similar to predicates, but define only intra-object characteristics rather than inter-object relationships. For example, every docset is defined to possess a `min_docs_requested` feature of type int. The rationale behind this representation was that it would reduce the complexity of the state representation: rather than explicitly enumerating all features of each object in the state, this construct enables them to be implicitly maintained and defined in terms of referring expressions. However, it should be noted that the current PLEXIS implementation doesn't correctly postulate features for projected objects (it is only aware of features of objects that actually exist). Consequently, this construct can only be used in the domain model if the planner executes after every planning step.

3.4 Metrics

In contrast to predicates and features, which define object-level relationships and characteristics, *metrics* define state-level system resources and information quality estimates based on all information objects present in the state. Their primary role is to serve as the arguments to the utility function used to estimate the state's information utility. The JAVELIN QA domain currently defines five metrics: SYSTEM_TIME, REQUEST_QUALITY, DOCSET_QUALITY, FILLSET_QUALITY, and ANSWER_QUALITY.

3.5 Operators

Operators define the set of QA processing actions that the Planner Module can control. As illustrated by the sample retrieval operator in Figure 7, each operator consists of *preconditions*, a set of *dynamic bindings*, a set of *probabilistic effects*, and an *execution specification*. Preconditions are logical expressions describing the predicates and metric value constraints that must hold before an operator is considered *applicable* in a state. Dynamic bindings define variables in the effects sets whose values are determined at run-time and depend on attributes of the state in which the action is applied. Probabilistic effects define all possible changes to the information state the operator may enact, in terms of its effects on predicates and metric values.

Currently, the Planner Module's domain defines a single operator for each QA component subsequent to question analysis, an additional check_answer operator used to compare the answer's confidence against a predefined confidence threshold, and three experimental operators to request feedback. The names of each operator are listed in Table 2. Our choice of operator set was driven primarily by an interest in evaluating system performance at the component-level. With this operator set, we can use the Planner Module to dynamically select one or more of the four extraction components using a model of their relative success rate for different types of questions.

It is important to recognize this is not the only set of operators that can be used to represent the JAVELIN system. We could choose to give the planner finer-grained control with operators corresponding to different parameter settings for a component (e.g., defining a separate retrieval operator for each different retrieval method available). We could also choose to give the planner less control by defining macro operators that call sequences of modules. Deciding what is appropriate depends primarily on: whether the operators result in different outcomes that we care about distinguishing between; and whether we can identify appropriate state features that reliably predict the context in which each operator should be used.

```
(:action RETRIEVE_DOCUMENTS
  :param (?q - question ?ro - qtype)
  :precond (and (request ?q ?ro)
           (not (no_docs_found ?ro))
           (not (exists (?d - docset)
              (retrieved_docs ?d ?ro)))
           (> (extracted_terms ?ro) 0)
           (> request_quality 0))
  :dbind (?docs (genDocsetID)
       ?dur (estTimeRS (expected_atype ?ro))
       ?pnone (probNoDocs ?ro)
       ?pgood (probDocsHaveAns ?ro)
       ?pbad (probDocsNoAns ?ro)
       ?dqual (estDocsetQual ?ro))
  :peffect (?pnone ((no_docs_found ?ro)
              (scale-down request_quality 2)
              (assign docset_quality 0)
              (increase system_time ?dur))
       ?pgood ((retrieved_docs ?docs ?ro)
             (assign docset_quality ?dqual)
             (increase system_time ?dur))
       ?pbad ((retrieved_docs ?docs ?ro)
            (scale-down request_quality 2)
            (assign docset_quality 0)
            (increase system_time ?dur)))
  :execute (RetrievalStrategist ?docs ?ro 10 15 300))
```

Figure 7: Sample document retrieval operator.

RETRIEVE_DOCUMENTS
EXTRACT_KNN_CANDIDATE_FILLS
EXTRACT_FST_CANDIDATE_FILLS
EXTRACT_LIGHT_CANDIDATE_FILLS
EXTRACT_SVM_CANDIDATE_FILLS
RANK_CANDIDATES
CHECK_ANSWERS
RESPOND_TO_USER
ASK_USER_FOR_ANSWER_TYPE
ASK_USER_FOR_MORE_KEYWORDS

Table 2: Summary of the operators currently used by the Planner Module to control the JAVELIN QA system.

3.6 Domain Functions for Operator Parameter Estimation

Dynamic bindings for variables in the operator effects are provided via a set of predefined *domain functions*. Prototypes of these functions are declared as part of the domain specification (loaded at run-time). The functions implementations themselves, however, must be provided when the Planner Module server is compiled.

Function prototypes declare argument and return types, and are used to perform simple type-checking of the domain operators that make use of them. All domain functions are currently implemented as C++ functions that inherit from a common DomainFunction class.

3.7 Problem Generation

For each new question it receives, the Planner Module must construct a new problem instance to seed the planning process. It does so by translating the output of the question analysis into a question-specific problem statement composed of: an *initial state*, a *utility function* $U(s)$, a minimum *goal utility threshold* G_{thresh} for successful termination, a minimum *satisfiability threshold* S_{thresh}, and an optional *symbolic goal condition* that must hold in the final state. The initial state defines the set of information objects that exist, the literals (instantiated predicates) that are currently true, and the initial values of each metric. The utility function defines the relative importance of each metric and resource, and is used to estimate progress towards the user's information goal. To be useful, the utility function must define values consistent with the user's underlying preferences, correctly mapping goal states to high utility values and non-goal states to low values.

Utility functions are defined in the PLEXIS domain language as weighted combinations of functions for individual metric values m in the domain, each of which produces a normalized value between zero and one:

$$U(s) = \frac{\sum_m w_m U_m(s)}{\sum_m w_m}$$

The utility threshold G_{thresh} specifies the minimum utility value required for a solution. Any executed sequence with a utility value greater than this is assumed to have achieved the goals. The satisfiability threshold defines how confident the planner must be that the current belief state actually satisfies the goal condition.

A sample problem statement is shown in Figure 8. The `:util-functions` declaration specifies which domain functions to call to produce normalized metric values, and the `:util` declaration determines the relative weight to assign to each constituent of the utility function. Currently, the Planner Module only knows how to generate problem statements that have this general form (e.g., it can only generate a single goal for a question; it does not know how to automatically decompose a question into multiple subgoals). The only components of the statement that vary from one question to the next are the literals and types of objects included in the initial state, and the threshold values to use.

```
(define (problem QA_test_problem)
  (:util-functions (QA_fn REQUEST_QUALITY)
            (RS_fn DOCSET_QUALITY)
            (RF_fn FILLSET_QUALITY)
            (AG_fn ANSWER_QUALITY)
            (ST_fn SYSTEM_TIME))

  (:objects Q0 - question
        RO0 - entity)

  (:init-state (1.0 (interactive_session)
            (request Q0 RO0)
            (expected_ans_format Q0 ranked)
            (SYSTEM_TIME 1.1)
            (REQUEST_QUALITY 0.4)
            (DOCSET_QUALITY 0.0)
            (FILLSET_QUALITY 0.0)
            (ANSWER_QUALITY 0.0))

  (:util (1 QA_fn)
     (3 RS_fn)
     (4 RF_fn)
     (6 AG_fn)
     (3 ST_fn))

  (:Sthresh 0.9)
  (:Gthresh 0.1)

  (:goal (exists (?a - atype ?al - anslist)
          (satisfies Q0 ?a ?al))))
```

Figure 8: Sample problem statement.

4 Communication

The Planner Module communicates with both the GUI and Execution Manager via TCP/IP sockets. This section describes the commands currently supported by the Planner Module and provides examples of their use. Formal specifications of all XML data can be found in Appendices B and C.

4.1 Server Protocol

The basic communication protocol used by the Planner, EM, and GUI consists of ASCII text messages prefixed by the number of bytes in the message and a single space:

<#bytes> <message>

The space serves as a delimiter and is not included in the byte count (e.g., the message 'OK' would be sent as '2 OK'). Each message consists of a single uppercase word denoting a command, possibly followed by a single space and plain text or XML data, depending on the command:

<message> := *<command>*(*<data>*)?

4.2 GUI-Planner Commands

Communication between the GUI and Planner Module consists of three types of exchanges: question processing, user feedback, and general process and planning task control. Table 3 summarizes the GUI commands currently recognized by the Planner and the contexts in which they are valid. Table 4 lists responses the Planner may return to the GUI and the contexts in which they occur. Many of these commands can be issued asynchronously (e.g., a single question command from the GUI typically receives multiple messages from the Planner during the course of generating an answer). Consequently, it is assumed that both the GUI and the Planner regularly poll their communication ports for new data.

COMMAND	DESCRIPTION	CONTEXT
QUESTION *<question XML>*	Initiate a question-answering session with the Planner	Valid when the Planner is not currently working on another question.
RESPONSE *<response text>*	Provide a response to a Planner dialog	Valid when the Planner has issued a dialog to the GUI and is waiting for a response.
PAUSE	Pause Planner processing	Valid at any point in the session.
RESUME	Resume the Planner processing when paused	Valid if the Planner is currently paused.
QUIT	Stop the current planning session	Valid at any point in the session.
STOP	Stop the current question-answering process (if any) and restore the Planner to the ready state (does not terminate the planning session)	Valid at any point in the session.
STATUS	Request Planner status and parameter settings.	Valid at any point in the session.
LOAD *<load xml>*	Change the current domain or problem in the Planner's working memory	Valid when the Planner is not currently working on a question.
RUN	Start planning and execution process on the domain and problem currently in working memory	Valid only when the Planner is not working on another question and when both a domain and problem have previously been loaded.

Table 3: GUI inputs supported by the Planner Module

COMMAND	DESCRIPTION	CONTEXT
OK	Acknowledge request receipt and initiation of processing	Issued in response to QUESTION, LOAD, and RUN requests, or PAUSE, RESUME and STOP control messages.
ERROR <*error text*>	Signal an error condition caused by invalid GUI input or a planning or execution failure (e.g., a module is down)	May be issued during an active question-answering session or in response to a GUI command.
STATUS <*status text*>	Displays Planner status and settings.	Issued in response to a STATUS request.
ANSWER <*answer XML*>	Return an answer to the GUI	Issued at the end of an active question-answering session. Returning this to the GUI entails the Planner is again available to service new requests.
DIALOG <*dialog XML*>	Request user input in the form of a yes/no, multiple-choice, or free text query	May be issued when a question-answering session is active and running in interactive mode.

Table 4: Responses and requests returned by the Planner Module to the GUI

Question Processing A sample question-answering exchange is illustrated in Figure 9. The GUI initiates a question-answering session with the Planner by issuing a 'QUESTION' command followed by XML data. The question XML contains the text of the user's question and optional user-defined values for several system parameters.[1] If the Planner is available to service the request, it will respond immediately with 'OK'. Otherwise, it will return an 'ERROR' command, followed by a plain text message describing the reason for failure. If the 'log' attribute is included in the request, the Planner will write (plain text) diagnostic messages to the specified host and port during the question-answering session. Otherwise, the Planner will not provide any diagnostic feedback to the GUI. The 'collection' argument enables the user to specify which document collection to search for an answer. If omitted, the default document collection of the RetrievalStrategist will be used. The two thresholds and time limit set termination criteria for the planner.

After a question request has been initiated and acknowledged, the Planner will construct and execute a plan to produce an answer. Upon completing this process, it returns an 'ANSWER' command followed by answer XML containing the data repository id assigned to the question and an ordered list of answers

[1] Besides those shown in the question example, several additional attributes (used for test-purposes) are recognized by the Planner. These attributes are documented in Appendix B.

```
GUI:      QUESTION <ANSWERQUESTION type ='new' interactive='true' log='128.2.111.11:1111'
            collection='TREC' utility-thresh='0.7' success-thresh='0.8'
            time='600'><![CDATA[What year did the Titanic sink?]]></ANSWERQUESTION>

Planner:  OK

Planner:  ANSWER <ANSWERLIST question_id="73945">
            <ANSWER id="1656111" confidence="0.75623">April 14, 1912</ANSWER>
            <ANSWER id="1656112" confidence="0.69612">May 27, 1941</ANSWER>
            <ANSWER id="1656113" confidence="0.28777">May 24, 1941</ANSWER>
            <ANSWER id="1656114" confidence="0.21414">October</ANSWER>
            <ANSWER id="1656115" confidence="0.19128">1985</ANSWER>
            <ANSWER id="1656116" confidence="0.10506">November</ANSWER>
            <ANSWER id="1656117" confidence="0.01094">July</ANSWER>
            <ANSWER id="1656118" confidence="0">1954</ANSWER>
          </ANSWERLIST>
```

Figure 9: Sample XML illustrating a GUI question request and subsequent responses returned by the Planner Module.

Planner:

```
DIALOG <DIALOG type='yes/no' default='no'>
  <QUESTION>
JAVELIN has interpreted your question as a request for a single answer.
Is this correct?
  </QUESTION>
</DIALOG>
```

GUI:

```
RESPONSE Yes
```

Planner:

```
<DIALOG type='multiple-choice' default='object'>
  <QUESTION>Please select the answer category that best matches the infor-
mation you are seeking.</QUESTION>
  <CHOICE>object</CHOICE>
  <CHOICE>temporal</CHOICE>
  <CHOICE>location</CHOICE>
  <CHOICE>proper-name</CHOICE>
  <CHOICE>person-name</CHOICE>
  <CHOICE>organization-name</CHOICE>
</DIALOG>
```

GUI:

```
RESPONSE temporal
```

Planner:

```
<DIALOG type='text'>
  <QUESTION>Current query terms are 'Titanic' 'sink'.  Please enter an
additional term (or leave blank if there are no additions).</QUESTION>
</DIALOG>
```

GUI:

```
RESPONSE iceberg
```

Figure 10: Examples of planner-initiated dialogs.

(possibly empty), their associated repository ids, and confidence scores. Return of an answer also signals that the Planner is again available to service new questions from the GUI.

Feedback If the question-answering session is running in interactive mode, the Planner may issue 'DIALOG' commands to the GUI, requesting user feedback (Figure 10). A dialog command is accompanied by XML data specifying the type and content of the feedback request. Supported types consist of yes/no questions, multiple-choice questions, and requests for textual input. Both yes/no and multiple-choice dialogs also provide default responses. The GUI displays the received dialog to the user, and returns the user's response to the Planner via a 'RESPONSE' command followed by the text of the user's reply.

Process Control 'PAUSE', 'STOP', and 'QUIT' commands will suspend the Planner, abort the current question process, or abort the session, respectively. The Planner Module will acknowledge 'PAUSE' and 'STOP' commands with 'OK' or return an 'ERROR' message if it is unable to comply with the request. When paused, the session can be resumed by sending a 'RESUME' command to the Planner. No acknowledgments or error messages are sent in response to a 'QUIT' request, and there are no provisions for resuming a stopped question process or terminated session. The GUI may also retrieve the server status and parameter settings at any point in time via the 'STATUS' command.

Planning Task Control 'LOAD' commands enable the GUI to change the current domain or problem in the Planner's working memory (Figure 11). The Planner reads in the domain or problem specification from the file provided in the load XML, and responds to the GUI with 'OK' or an 'ERROR' message if it cannot complete the request. The Planner first looks for the filename as given, but failing that, it also looks for a

```
     GUI:   LOAD <DOMAINFILE>/usr0/javelin/sandbox/javelin/planner/server/QA.domain</DOMAINFILE>

 Planner:   OK

     GUI:   LOAD <PROBLEMFILE>Trec_q1.problem</PROBLEMFILE>

 Planner:   OK

     GUI:   RUN

 Planner:   OK
```

Figure 11: Loading and running a planning domain and problem.

file of that name with respect to the default domain search path $JAVELIN_ROOT/planner/domains. If both a domain and problem are currently defined, the Planner can then be invoked using the 'RUN' command.[2]

4.3 Planner-EM Commands

The Planner calls the EM to execute specific actions in the QA process, to modify information objects, and to store planning data in the repository for later use. Unlike communication between the GUI and Planner Module, communication between the Planner and EM always consists of single request-response pairs. Table 5 summarizes the commands used by the Planner for these tasks, along with the replies it expects the EM to return.

Session ID Management At any given point in time, there may be multiple copies of the Planner server sharing an Execution Manager and QA system. In order to ensure a unique correspondence between repository ids and planner ids of information objects, the Planner obtains a unique numeric session id from the

[2] Although implemented, the 'RUN' command is not functional because the Planner Module does not currently support execution initiated with data read from a static problem file, only via problem statements dynamically generated from question input.

COMMAND	DESCRIPTION	EM RESPONSE
GETID	Request a new session id from the repository	NEWID <*planner id XML*> or ERROR <*error text*>
EXECUTE <*execution XML*>	Call one of the QA modules using the specified data as input	RESULTS <*results XML*> or ERROR <*error text*>
STORE <*planner data XML*>	Save planning state information in the repository	OK or ERROR <*error text*>
MODIFY <*modification XML*>	Modify/update an information object in the repository	OK or ERROR <*error text*>
BATCH <*batch request XML*>	Perform a data storage operation related to a Planner batch test	SAVED <*batch data XML*> or ERROR <*error text*>

Table 5: Commands issued by the Planner Module and corresponding responses returned by the Execution Manager.

Planner:	`GETID`
EM:	`NEWID <PlannerID id="2331">`

Figure 12: A Planner request for a session id.

EM at the start of each planning session, which is then included on all data the Planner produces during the session. Within a session, the Planner is responsible for ensuring any information object ids it generates are unique, but these planner-generated ids need not be unique between sessions. We define a *planning session* as any sequence of planning and execution steps performed by a single Planner instance that we wish to group together (for a typical trivia-type question, this translates to a new session id for each question, while both complex and contextual questions will typically have multiple questions associated with a single session). The Planner obtains this session ID by issuing a 'GETID' command, to which the EM responds with 'NEWID' followed by XML containing the numeric identifier (Figure 12).

Execution Control The Planner maintains an abstract representation of the information state and is unaware of the details required to actually call the individual QA modules. To execute an action, the Planner relies on the Execution Manager to reconstruct the input required by the QA module, supplying it only with the planner repository ids for the information objects to use as inputs, a new planner repository id for each information object the planner expects the execution to create, and module-specific arguments and constraints (e.g., upper bounds on execution time, the minimum and maximum number of documents to retrieve). Once the EM has called the appropriate QA module and received a response, it will construct a reply to send back to the Planner using a 'RESULTS' message followed by XML data. With the exception of output from calls to the AnswerGenerator, the results XML is just the raw XML output produced by the QA module, wrapped within a pair of 'Results' tags and annotated with the EM's processing time (in seconds), plus the session and exe_id of the EXECUTE request it is in response to. AnswerGenerator XML output is modified to include the data repository ids assigned to each answer.[3] If an error condition occurs (e.g., the requested QA module does not respond), then the EM returns an 'ERROR' message followed by a text description of the failure. Note that error conditions do not include module exceptions, which are treated as results.

Figure 13 illustrates a sample execution call to the document retrieval module (RS) and the response returned by the Execution Manager. The Planner has instructed the EM to call the RS using the RequestObject with a planner id of 'RO4695' as input. It has also specified that the RS should retrieve between 10-15 documents from the 'AQUAINT' collection within a time limit of 300 seconds. The DocumentSet produced by this call will be saved in the system repository by the EM under planner id 'DS4475'.

Data Storage The Planner issues 'STORE' requests to save three types of planning process information in the Repository: the initial planning problem, the outcome of a planning step (any new candidate actions that are generated and the action the Planner has chosen to add to the current partial plan), and outcomes of executing actions in the plan. Each is formatted in XML distinguished by the outermost tags: initial state information is contained within 'InitialState' tags, planning step information is contained within 'PlanningStep' tags, and execution outcome information is contained within 'ExecutionOutcome' tags. Examples of each are presented in Figures 14 through 16.

[3] The Planner passes this information along with any answers it returns to the GUI, enabling the GUI to create hyperlinks between the answers and their source.

```
Planner:  EXECUTE <Execute version="0.3" exe_id="17975" session_id="6044">
            <Command name="RetrievalStrategist">
              <Assigns object="DocumentSet">DS4475</Assigns>
              <Arg name="Collection">AQUAINT</Arg>
              <Arg name="Maxdoc">15</Arg>
              <Arg name="Mindoc">10</Arg>
              <Arg name="RequestObject">RO4695</Arg>
              <Arg name="Time">300</Arg>
            </Command></Execute>

     EM:  RESULTS <Results version="0.3" exe_id="17975" session_id="6044" EM_time="1">
            <RetrievalStrategist version="2.1" status="OK">
            <ResourceStats>
              <Time unit="sec">6.55</Time>
            </ResourceStats>
            <RequestObject id="15552"/>
            <Constraints>
              <Source>AQUAINT</Source>
              <Mindoc>10</Mindoc>
              <Maxdoc>15</Maxdoc>
            </Constraints>
            <DocumentSet>
              <Document source="AQUAINT" trecID="NYT19990721.0145" docID="405900"
               score="0.52831">
                <Query>#UW10( book #3( rachel carson ) 1962 write *work_of_art )</Query>
              </Document>
              <Document source="AQUAINT" trecID="NYT19990811.0149" docID="413653"
               score="0.605419">
                <Query>#UW10( #3( rachel carson ) 1962 write *work_of_art )</Query>
              </Document>
              <Document source="AQUAINT" trecID="NYT19990901.0198" docID="421374"
               score="0.507894">
                <Query>#UW10( #3( rachel carson ) 1962 write *work_of_art )</Query>
              </Document>
          ...
              <Document source="AQUAINT" trecID="NYT19991230.0073" docID="462648"
               score="0.492014">
                <Query>#UW10( #3( rachel carson ) 1962 )</Query>
              </Document>
            </DocumentSet>
            </RetrievalStrategist></Results>
```

Figure 13: A call to retrieve documents from the RetrievalStrategist. Results have been truncated to conserve space.

Object Modification To modify an information object (e.g., to revise the contents of a document set), the Planner issues a 'MODIFY' request accompanied with XML describing the object to be changed, and the change to make (Figure 17). This function is used when we have additional information, such as feedback from the user, that must be incorporated into the information state. All modifications are made by cloning the objects to maintain traceability in the Repository.

Planner:

```
STORE <InitialState version="0.1" question_id="15553" session_id="6044">
  <Action id="A0">
   <![CDATA[INITIALIZE QuestionAnalyzer RO4695 180
      'What book did Rachel Carson write in 1962?']]>
  </Action>
  <BeliefState id="B89">
    <State id ="S202" prob="1" util="0.130831">
    <MetricSet>
      <Metric name="ANSWER_QUALITY" value="0"/>
      <Metric name="DOCSET_QUALITY" value="0"/>
      <Metric name="FILLSET_QUALITY" value="0"/>
      <Metric name="REQUEST_QUALITY" value="0.4"/>
      <Metric name="SYSTEM_TIME" value="15.765"/>
    </MetricSet>
    <Objects>RO4695:entity,NEW:context,RANKED:aformat,Q15553:question,...,
     DICT:ix-name,DT:ix-name,FST:ix-name,KNN:ix-name,LIGHT:ix-name,SVM:ix-name
    </Objects>
    <Literals>(expected_ans_format RANKED),(request Q15553 RO4695)</Literals>
    </State>
  </BeliefState>
  <Goal Gthresh="0.1" Sthresh="0.1">(exists ( ?a:atype ?al:anslist)
   (satisfies Q15553:question ?a:atype ?al:anslist))</Goal>
  <UtilityFunction>
    <Function name="AG_fn" param="ANSWER_QUALITY" weight="0.333333"/>
    <Function name="QA_fn" param="REQUEST_QUALITY" weight="0.0952381"/>
    <Function name="RF_fn" param="FILLSET_QUALITY" weight="0.285714"/>
    <Function name="RS_fn" param="DOCSET_QUALITY" weight="0.190476"/>
    <Function name="ST_fn" param="SYSTEM_TIME" weight="0.0952381"/>
  </UtilityFunction>
</InitialState>
```

EM: `OK`

Figure 14: Sample InitialState XML. Object fields are truncated to conserve space.

Planner:

```
STORE <PlanningStep version="0.1" session_id="6044">
  <CandidateAction applicable_in="B89" EU="0.176704">
    <Action id="A61">RETRIEVE_DOCUMENTS
     RetrievalStrategist DS4475 RO4695 10 15 300</Action>
    <BeliefState id="B90">
     <State id ="S203" prob="0.2" util="0.110037">
      <MetricSet>
       <Metric name="ANSWER_QUALITY" value="0"/>
       <Metric name="DOCSET_QUALITY" value="0"/>
       <Metric name="FILLSET_QUALITY" value="0"/>
       <Metric name="REQUEST_QUALITY" value="0.2"/>
       <Metric name="SYSTEM_TIME" value="26.765"/>
      </MetricSet>
      <Objects>RO4695:entity,NEW:context,RANKED:aformat,...,Q15553:question,
       DS4475:docset,DICT:ix-name,DT:ix-name,FST:ix-name,KNN:ix-name,
       LIGHT:ix-name,SVM:ix-name</Objects>
      <Literals>(expected_ans_format RANKED),(request Q15553 RO4695),
       (retrieved_docs DS4475 RO4695)</Literals>
     </State>
     <State id ="S204" prob="0.7" util="0.205275">
      <MetricSet>
       <Metric name="ANSWER_QUALITY" value="0"/>
       <Metric name="DOCSET_QUALITY" value="0.4"/>
       <Metric name="FILLSET_QUALITY" value="0"/>
       <Metric name="REQUEST_QUALITY" value="0.4"/>
       <Metric name="SYSTEM_TIME" value="26.765"/>
      </MetricSet>
      <Objects>RO4695:entity,NEW:context,RANKED:aformat,...,Q15553:question,
       DS4475:docset,DICT:ix-name,DT:ix-name,FST:ix-name,KNN:ix-name,
       LIGHT:ix-name,SVM:ix-name</Objects>
      <Literals>(expected_ans_format RANKED),(request Q15553 RO4695),
       (retrieved_docs DS4475 RO4695)</Literals>
     </State>
     <State id ="S205" prob="0.1" util="0.110037">
      <MetricSet>
       <Metric name="ANSWER_QUALITY" value="0"/>
       <Metric name="DOCSET_QUALITY" value="0"/>
       <Metric name="FILLSET_QUALITY" value="0"/>
       <Metric name="REQUEST_QUALITY" value="0.2"/>
       <Metric name="SYSTEM_TIME" value="26.765"/>
      </MetricSet>
      <Objects>RO4695:entity,NEW:context,RANKED:aformat,...,Q15553:question,
       DICT:ix-name,DT:ix-name,FST:ix-name,KNN:ix-name,LIGHT:ix-name,
       SVM:ix-name</Objects>
      <Literals>(expected_ans_format RANKED),(request Q15553 RO4695),
       (no_docs_found RO4695)</Literals>
     </State>
    </BeliefState>
  </CandidateAction>
  <Outcome status="OK" addedToPlan="A61"/>
</PlanningStep>
```

EM: OK

Figure 15: Sample PlanningStep XML. Object fields are truncated to conserve space.

Planner:

```
STORE <ExecutionOutcome version="0.1" exe_id="17975" applied_to="B89" ses-
sion_id="6044">
  <Action id="A61">RETRIEVE_DOCUMENTS RetrievalStrategist DS4475 RO4695 10 15 300</Action>
  <BeliefState id="B91">
   <State id ="S206" prob="1" util="0.225029">
    <MetricSet>
      <Metric name="ANSWER_QUALITY" value="0"/>
      <Metric name="DOCSET_QUALITY" value="0.5"/>
      <Metric name="FILLSET_QUALITY" value="0"/>
      <Metric name="REQUEST_QUALITY" value="0.4"/>
      <Metric name="SYSTEM_TIME" value="22.315"/>
    </MetricSet>
    <Objects>RO4695:entity,NEW:context,RANKED:aformat,...,Q15553:question,
     DS4475:docset,DICT:ix-name,DT:ix-name,FST:ix-name,KNN:ix-name,LIGHT:ix-name,
     SVM:ix-name</Objects>
    <Literals>(expected_ans_format RANKED),(request Q15553 RO4695),
     (retrieved_docs DS4475 RO4695)</Literals>
   </State>
  </BeliefState>
</ExecutionOutcome>
```

EM: OK

Figure 16: Sample ExecutionOutcome XML. Object fields are truncated to conserve space.

Planner:

```
MODIFY <ObjectModification version="0.3" session_id="111">
  <ObjectToUpdate type="RequestObject" id="RO452" newid="RO453">
    <Replace><AnswerType confidence="0.9">numeric</AnswerType></Replace>
    <Replace><QuestionType confidence="0.9">entity</QuestionType></Replace>
    <Remove><Keyword type="word">inhabitants</Keyword></Remove>
    <Add><Keyword type="word">people</Keyword></Add>
    <Require><Keyword type="proper">Ushuaia</Keyword></Require>
  </ObjectToUpdate>
</ObjectModification>
```

EM: OK

Planner:

```
MODIFY <ObjectModification version="0.3" session_id="111">
  <ObjectToUpdate type="DocumentSet" id="DS123" newid="DS124">
    <Replace><Document trecID="FT922-7671" confidence="1.0" /></Replace>
    <Remove><Document trecID="FT933-12217" confidence="0.458005" /></Remove>
  </ObjectToUpdate>
</ObjectModification>
```

EM: OK

Planner:

```
MODIFY <ObjectModification version="0.3" session_id="111">
  <ObjectToUpdate type="RequestFillSet" id="FS234" newid="FS235">
    <Replace><RequestFill id="1234" confidence="0.9" /></Replace>
    <Remove><RequestFill id="1235" confidence="0.223" /></Remove>
  </ObjectToUpdate>
</ObjectModification>
```

EM: OK

Figure 17: Sample object modification XML illustrating changes to a RequestObject, DocumentSet, and RequestFillSet.

```
Planner:   BATCH <BatchRequest version="0.1">
                   <Command name="Initialize">
              <Arg name="TestCategory">Location</Arg>
           </Command>
                 </BatchRequest>

     EM:   SAVED <BatchData version="0.1">
                   <BatchDir>/usr0/htdocs/javelin/Planner/0402\_601</BatchDir>
                 </BatchData>

Planner:   BATCH <BatchRequest version="0.1">
                   <Command name="StartQuestion">
                     <Arg name="TrecID">111</Arg>
                   </Command>
                 </BatchRequest>

     EM:   SAVED

Planner:   BATCH <BatchRequest version="0.1">
                   <Command name="EndQuestion"></Command>
                 </BatchRequest>

     EM:   SAVED <BatchData version="0.1">
                   <CachedXMLs>
                     <File>QA\_Input\_q111\_ro12345.txt</File>
                     <File>QA\_Output\_q111\_ro12345.txt</File>
                     ...
                   </CachedXMLs>
                 </BatchData>

Planner:   BATCH <BatchRequest version="0.1">
                   <Command name="Terminate">
                     <Arg name="MRR">0.25</Arg>
                     <Arg name="TrecScore">0.15</Arg>
                     <Arg name="QuestionFile">questions.PL001</Arg>
                     <Arg name="DomainFile">QA.domain.PL001</Arg>
                     <Arg name="Description"><![CDATA[Test description]]></Arg>
                   </Command>
                 </BatchRequest>

     EM:   SAVED
```

Figure 18: Sample batch test exchanges with the EM illustrating test initialization, the start and end of a question within the test, and test termination.

Batch Test Data Storage The Execution Manager handles Repository storage of Planner batch test results and local caching of data created during these tests via the 'BATCH' command. Each 'BATCH' command is issued with a single XML-formatted request. There are currently four types of batch requests recognized by the EM: test initialization, test termination, and requests signaling the start and end of individual questions within the batch test (Figure 18). The EM will respond to these commands with a 'SAVED' message if the request was successfully carried out, or (as with all other commands) an 'ERROR' message if an unexpected error occurs while processing any of the BATCH requests.

An 'Initialize' request is sent at the start of each batch test, and indicate the Execution Manager should start caching all of the input and output produced in subsequent calls to individual QA modules. The EM also takes care of assigning a new batch id to this test, and creates the directory where the input and output files will be saved.[4] The EM uses the 'TestCategory' argument to determine where the new directory should be

[4]The root of the XML cache directory path used by the EM is determined by the `LogFileDir` variable in the `ExecutionManager.properties` file and can only be changed by modifying the property file and restarting the EM server.

created in the JAVELIN web site (e.g., under the set of "Location" tests or under the "Planner" test category). If the EM successfully completes these initialization steps, it will respond with a 'SAVED' message followed by BatchData XML specifying the absolute path of the cached XML directory (this directory information is used to construct links in the html file produced at the end of the batch test).

A 'StartQuestion' request is sent before each question in the batch test set. It provides the EM with the TREC id of the current question, which is then included in the names of all cached XML data files the EM saves for that question. The EM acknowledges receipt of this id by returning a 'SAVED' message without any XML data. A corresponding 'EndQuestion' request is issued immediately after processing the question, indicating the EM should return a list of all the XML data files it saved for the most recently processed question.

After the last question of the batch test has been processed, a 'Terminate' request is sent to the Execution Manager. In addition to signaling the EM to stop caching module data, it also provides the EM with the mean reciprocal rank (MRR) and accuracy (referred to as the 'TREC score') computed for the test, the names of the question and planning domain files used by the test, and an optional test description. This information is saved by the EM in the Repository and used to update the Planner test results table maintained on the JAVELIN web site. Successful termination of the batch test is indicated by the return of a 'SAVED' message.

It should be noted that while batch processing support is part of the Planner-EM communication protocol, the current implementation of the Planner Module has no knowledge of or support for the 'BATCH' command and its responses. All batch commands are issued by a meta-level perl script that controls the batch test and interacts with the Planner Module to simulate GUI behavior (see Section 5.6.4 for details).

5 Installation and Execution Instructions

5.1 CVS Directory Organization

All of the Planner Module source code and test scripts reside within the `planner` subdirectory of the main JAVELIN CVS directory. This directory is organized into the following six subdirectories:

- **`engine`** - C++ source files implementing the planning functionality
- **`server`** - C++ source files implementing the server, JAVELIN-specific and QA-domain-specific functions (e.g. XML parsing, translation between the QA system and internal planner data representations)
- **`etc`** - general run-time data files (i.e., a default `config` file)
- **`domains`** - sample planning domain and problem specification files
- **`tools`** - perl modules, miscellaneous perl scripts for debugging batch test results, TREC datasets (questions, answer patterns and document judgements)
- **`test`** - perl implementations of the AnswerOracle and DomainTranslator submodules, plus perl scripts for run-time testing

5.2 Compiling the Planner Module Server

The Planner Module is written in C++, and has been compiled and tested with Linux RedHat 7.1 using g++ version 3.0.1. It requires the Xerces C++ library (version 2.5.0) for XML parsing support, and uses Flex++ and Bison to build the domain parser used by the planner.

The build process is controlled by a `makefile` in the `planner` directory, which in turn depends on `Makefile.common` in the top-level `javelin` directory. You must edit the Planner Module makefile to reflect the local source and library paths for the machine you are using. Compilation and installation of the Planner Module server can then be initiated using '`make build`' and '`make deploy`' commands, respectively. By default, the resulting executable and all supporting run-time scripts and default configuration files are placed in the `deploy/planner` directory of the `javelin` directory.

5.3 Creating a Configuration File

At run-time, the Planner Module loads its default settings from a configuration file. By default, the Planner searches for a file named `config` in the same directory as the server executable. Alternatively, a configuration filename can be supplied as a run-time argument when the server is started.

A sample configuration file is provided in Figure 19. Each line consists of a single parameter name and its corresponding default value, separated by whitespace. Text beginning with a '#' character is treated as a comment and ignored (up to the end of the current line), as are lines consisting solely of whitespace characters. All of the parameters related to server and submodule settings can only be altered at server startup by changing this configuration file. However, several of the parameters defining planner and GUI defaults may be overridden subsequently on a per-question basis by providing new values as part of the question processing request. A detailed summary of the parameters currently recognized by the Planner Module and their use is provided in Table 6.

```
# Server and submodule settings
# -----------------------------
Port 2003
LogFile server.log
LogLevel 3
EMHost localhost:2002
AnswerOracleHost localhost:2011
DomainTranslatorHost localhost:2022
UseOracleFor none

# Planner defaults
# ----------------
DomainDirDefault /usr0/javelin/sandbox/javelin/planner/domains
DataDirDefault /usr0/javelin/sandbox/javelin/planner/domains/data
DomainDefault QA.domain
DomainParamDefault QA.params
GthreshDefault 0.1
SthreshDefault 0.1
TimeDefault 600
ExecutionStrategy RES
ReplanningStrategy always
StoppingCriteria nop

# GUI interaction defaults
# ------------------------
InteractiveDefault false
AnswerFormatDefault ranked
AnswerLength short
AnswerMaxCount 30
AnswerMinUtil 0.05
```

Figure 19: Sample Planner Module configuration file.

Parameter	Possible Values	Description
Port	*integer between 2000 and 9999*	Port where the Planner Server will reside. The standard JAVELIN configuration expects the Planner Module to use port 2003.
LogFile	*filename*	File to which the Planner Module will write log message.
LogLevel	*integer between 1-3*	Controls log message verbosity (3=high)
EMHost	*<hostname>:<port>*	Host and port for the Execution Manager.
AnswerOracleHost	*<hostname>:<port>*	Host and port for the AnswerOracle submodule.
DomainTranslatorHost	*<hostname>:<port>*	Host and port for DomainTranslator submodule.
UseOracleFor	none|QA|RS|IX|AG|all	Specifies which modules' output should be corrected by the AnswerOracle prior to passing the results to the planner.
DomainDirDefault	*unix-style directory (absolute path)*	Default domain directory to search for planning domain and problem specification files.
DataDirDefault	*unix-style directory (absolute path)*	Default directory used by the DomainTranslator to store/retrieve execution results for operator parameter estimation.
DomainDefault	*filename*	Default planning domain file to load.
DomainParamDefault	*filename*	Default file containing domain parameter models (see `DomainFunctions.cpp` and `DomainParameterTable.cpp`)
GthreshDefault *	*float between 0 and 1*	Default utility threshold to use during dynamic planning problem generation.
SthreshDefault *	*float between 0 and 1*	Default satisfiability threshold to use during dynamic planning problem generation.
TimeDefault *	*positive integer*	Default time limit to assign to a planning and execution session.
ExecutionStrategy	*see* `PlannerSettings.h` *for options*	Sets the Planner's execution strategy.
ReplanningStrategy	*see* `PlannerSettings.h` *for options*	Sets the Planner's replanning strategy.
StoppingCriteria	*see* `PlannerSettings.h` *for options*	Sets the Planner's failure criteria.
InteractiveDefault *	true | false	Default setting to enable/disable user-interaction during planning.
AnswerFormatDefault	ranked | single	Default results format to use when returning answer(s) to the GUI.
AnswerLength	short | long	How much supporting information to display with each answer.
AnswerMaxCount	*positive integer*	Maximum number of answers to display (relevant only for ranked answer format).
AnswerMinUtil	*float between 0 and 1*	Minimum utility value for selected/displayed answers.

Table 6: Configuration parameters recognized by the Planner Module. * denotes parameters that can be overridden for an individual request.

5.4 Running the Planner Module Server

Once you have set the `LD_LIBRARY_PATH` environment variable to include the Xerces C++ library directory and revised the Planner configuration file to reflect your local installation, the server may be started from the command line by typing:

```
./plannerRuntime config &
```

Alternatively, the server can be started using the `run_planner` script. This shell script is automatically created in the `deploy/planner` directory during the build process, and takes care of setting both the path and starting the server process.

5.5 Troubleshooting

The Planner Module provides both exceptions and a logging utility to help the user identify the root cause of processing failures.

5.5.1 Exceptions

Run-time errors in the Planner Module are signaled by five top-level exceptions: ServerFailureExceptions, JAVELINSocketExceptions, ConfigurationExceptions, QAExceptions, and PlannerExceptions. A ConfigurationException is thrown when the server is unable to parse the configuration file at startup. A JAVELINSocketException signals an error with the TCP/IP socket communication, and a ServerFailureException is thrown when some other unrecoverable error occurs within the server code. A QAException indicates a problem related to interaction with the GUI, Execution Manager, or QA data processing. A PlannerException signals a failure within the planner itself, typically arising from errors in the domain or problem specifications. Both QAExceptions and PlannerExceptions encompass more specific exception subcategories, the details of which can be found in the `QAExceptions.h` and `PlannerExceptions.h` header files, respectively.

5.5.2 Logfiles

Generally, all errors resulting in an exception are also recorded in the server log file. However, by setting the LogLevel parameter to its highest value, the log file can also be used to trace the planning session and intermediate data results. Each entry in the log is labeled with a timestamp and the name of the source file from which the log message originated. A sample excerpt is shown in Figure 20.

5.6 Supplemental Test and Evaluation Scripts

All of the perl scripts described in this section reside in the `test` subdirectory of the `planner` CVS subtree. Supporting JAVELIN-specific perl modules used by these scripts can be found in the `tools` subdirectory.

5.6.1 `plannerClient.pl`: A Command-line Planner Client

This perl script enables a user to interact with the Planner Module from the command line. The script is invoked with two arguments specifying the host and port of the Planner Module you wish to communicate with, e.g.:

```
./plannerClient.pl orissa 2003
```

After starting the script, any of the GUI commands (defined previously in Table 3) may be sent to the Planner Module by typing the command at the prompt and pressing `[RET]`. Responses received from the Planner will be printed to stdout. The script can be terminated by sending an empty message (hitting `[RET]` at the prompt), or by typing either `quit` or `exit` followed by `[RET]`.

```
PLServer [5/4/2004 15:51:05] *** PlannerModule 2.0 May  4 2004 13:45:54 (g++ 3.0.1) ***
PLServer [5/4/2004 15:51:05] [Server process 5823 started]
PLServer [5/4/2004 15:51:13] [Child process 5832 started]
PLServer [5/4/2004 15:51:16] Received client request:
QUESTION <ANSWERQUESTION interactive='f'>Where is Big Ben?</ANSWERQUESTION>
GUIDataTypes [5/4/2004 15:51:16] Parsing the GUI request params...
GUIDataTypes [5/4/2004 15:51:16] Created request:
[ REQUEST Where is Big Ben?
  trecID:
  context: new
  collection:
  amount constraint:
  atype constraint:
  interactive: 0
  log: 0
  util: 1
  succ: 0.5
  time limit: 600 ]

EMInterface [5/4/2004 15:51:16] Calling the EM with: GETID
PLServer [5/4/2004 15:51:16] Requesting new session id from EM
EMInterface [5/4/2004 15:51:16] Calling the EM with:
EXECUTE <Execute version="0.3" exe_id="1498" session_id="44444">
<Command name="QuestionAnalyzer"><Assigns object="RequestObject">RO300</Assigns>
<Arg name="Question"><![CDATA[Where is Big Ben?]]></Arg>
<Arg name="Time">120</Arg></Command></Execute>
EMDataTypes [5/4/2004 15:51:16] Parsing EM response...
EMDataTypes [5/4/2004 15:51:16]  EM time: 0.2
XMLTools [5/4/2004 15:51:16] Reading question type
XMLTools [5/4/2004 15:51:16] Reading answer type(s)
XMLTools [5/4/2004 15:51:16] Reading keyword(s)
QADataTypes [5/4/2004 15:51:16] [ RequestObject
  questionID: Q1671
  Qtype: entity (0.9)
  Atype: location (1)
  parent:
  super:
  order:
  qty: 1
  eval:
  terms: 'Big Ben' ]

XMLTools [5/4/2004 15:51:16] Reading module execution time
XMLTools [5/4/2004 15:51:16]  Elapsed time: 100
DomainTranslator [5/4/2004 15:51:16] Completed planner problem creation
```

Figure 20: Excerpt from the Planner Module log file.

5.6.2 `dummyEMServer.pl`: An EM Server Based on Cached XML

This perl script implements a simple server that mimics the EM server behavior by returning cached XML output from previous runs on the TREC question sets. It is intended for use by the planner server during debugging (to avoid time delays associated with QA module execution, and to enable continued development work when any of the system components are unavailable.)

It can only be used with the TREC question sets, and all test questions must be typed exactly as they appear in the TREC question files. (It relies on a simple text match with the TREC questions to look up the TREC id, which in turn is used to retrieve the cached XML files for that question.) Moreover, since the script relies on cached XML to recreate the outputs, you can only use it to rerun the same sequence of

EXECUTE requests used to create the XML initially; you cannot use it to create new sequences or data. The script also does not include repository support. It will gracefully handle any STORE or MODIFY request by returning "OK", but the requests themselves are simply ignored. As with the real EM server, only one client request is processed per connection (i.e., the server always disconnects after processing a request).

The script takes one optional argument specifying the port to start the server on. If no port is specified, it will attempt to use port 2002. Before running this script, you must modify the variables specifying the location of the cached XML output for the individual modules, and the directory containing the individual questions split by answer-type (e.g., `$JAVELIN_ROOT/em/test/trec/`). The script assumes that each question file uses the naming convention <*type*>`.list`. You also may need to modify the setSubDir() subroutine within the perl script.

5.6.3 `answerOracle.pl`: A Submodule for Controlled Evaluation of Planner Behavior

This perl script provides a server (submodule) the Planner Module may call to repair select features and confidence scores of the data objects produced during the QA process. Its purpose is to enable a developer to perform controlled studies of the Planner behavior using the TREC questions: to confirm the Planner does the right thing (in an algorithmic sense) for different confidence/quality score distributions, to study how sensitive the planning process is to perturbations in these distributions, and to provide feedback on the degree of disparity between "ideal" scores and the "real" scores the QA components produce.

Currently, the oracle assigns scores to each document and answer candidate using the NIST document judgements and answer patterns. Confidence scores are assigned to question and answer types based on files containing manually assigned answer type classifications and correspondence maps between the question types and answer types. The supported oracle commands and rules used by the oracle to assign scores in response to each command are listed in Table 7. Note that the vertical bar character '|' is interpreted by the server as an argument delimiter in multi-argument commands.

The script is invoked by typing:

```
./answerOracle.pl [-I <judgement dir>] [<port>]
```

The two optional arguments enable the user to specify the port to start the server on, and whether to operate the oracle in an "interactive" mode. If the server is started in interactive mode, then requests that would receive a score of 0.5 using the default oracle behavior will instead be presented to the user to judge. These judgements are saved in the directory specified, and added to the working set of judgements.

5.6.4 `batchPlannerTest.pl`: Batch Test Support for TREC Question Evaluation

This script enables execution of the TREC questions in batch mode. It interacts with both the Planner and the EM to control question execution, and automatically generates two files: a plain-text summary of the results (`BatchLog_summary.txt`) and a more detailed log file in html (`BatchLog.html`). The script is invoked at the command line by typing:

```
./batch_planner_test.pl [-c] [-k] [-t <test type>] <PL host> <PL port> \
<EM host> <EM port> <question file> <domain file>  \
<PL server log> ['<description>']
```

For example,

```
./batch_planner_test.pl -c -t Trec9  orissa 2003 orissa 2002 \
  trec9_main_questions.txt QA.domain server.log 'Trec9 Qs' &
```

The first two arguments specify the host and port of the Planner Module, followed by arguments specifying the host and port of the EM server that will be used by the Planner, the file containing the set of TREC questions to run, the name of the planning domain file in use, the name of the Planner Module's logfile, and a brief description (optional) of the test to be run, enclosed in single quotes. The optional '`-c`' flag indicates the resulting html file should be color-coded, the '`k`' flag indicates the answer patterns developed by the JAVELIN team should be used to evaluate performance (the default is to use the NIST-supplied patterns), and the '`-t` <*test type*>' option indicates how the EM should classify the test (the default classification is 'Planner').
The assumed input format for each line of the question file is:

Command	**Response**	**Generated When**
TRECID <*question text*>	VALUE <*trecID*>	The question is identified as a TREC question.
	VALUE	All other cases.
DOCUMENT <*ext_docID*>|<*trecID*>	VALUE 1	The document is included in NIST's list of relevant documents for <*trecID*>.
	VALUE 0.5	The document body matches the answer pattern for <*trecID*>.
	VALUE 0	All other cases.
CANDIDATE <*candidate*>|<*trecID*>	VALUE 1	The TREC answer pattern for <*trecID*> matches the candidate (including inexact matches).
	VALUE 0	All other cases.
ANSWER <*answer*>|<*trecID*>	VALUE 1	The TREC answer pattern for <*trecID*> matches the answer exactly.
	VALUE 0.5	The answer is an inexact match with the answer pattern for <*trecID*>.
	VALUE 0	All other cases.
QTYPE <*qtype*>|<*trecID*>	VALUE 1	The answer type for <*trecID*> has a corresponding question type of <*qtype*>, as defined in the `qtype.map` file.
	VALUE 0	All other cases.
ATYPE <*atype*>|<*trecID*>	VALUE 1	The answer type for <*trecID*> matches <*atype*> (either exactly or via a correspondence defined in the `atype.map` file),
	VALUE 0	All other cases.
any of the above	ERROR <*error text*>	The oracle receives a request it cannot process or an internal error occurs.

Table 7: Commands recognized by the oracle, possible responses, and their corresponding score generation rules.

<trecID>:*<spc><question text>*

This script ***must*** be run on the same machine as the Planner Module it calls, and the Planner must be running with logging set to the highest level, as the batch script relies on the log's contents to extract the document sets and execution times for each question. It also assumes that the active planning domain does not contain any interactive operators.

At the end of the batch test, the script will try to copy the two summary files it generates to the batch directory directory created by the EM. This step will only succeed when the EM also resides on the same machine as the script. If the copy fails, the resulting `BatchLog*` files must be manually copied from the script's invocation directory to the EM's batch archive directory.

6 Discussion and Future Research Directions

The long-term goal of the JAVELIN research is to provide a flexible QA architecture that enables advanced question-answering tasks including: automated question decomposition and answer synthesis, knowledge reuse, user-interaction, and context-sensitive QA. To this end, we have described our initial implementation of the JAVELIN Planner Module, designed with these challenges in mind. Our choice of a utility-based planning paradigm is motivated by the need to support partial-satisfaction of information goals, both singly and within the context of planning for multiple question subgoals, as well as the need to provide strategies that are sensitive to the question context and a user's preferences. Our representation of the QA process reflects the need to decouple the common elements of the QA task from non-essential details of individual questions.

The Planner Module was successfully used to control the JAVELIN QA system during the TREC 2003 QA evaluation [4]. Although we did not spent much time tuning the planning parameters or operator models, the planner-based system achieved accuracy comparable to the single-best strategy without the Planner Module, and provided more robust handling of unexpected errors in redundant components such as the extractors.

However, much work remains if we are to actually realize the long-term goals. Future directions include:

- developing better models of the QA components' performance, including models of their execution time, and integrating a learning component into the Planner Module
- supporting sequences of planning requests in which multiple questions have a shared planning session and information context
- providing support for automated question decomposition
- better support for user-feedback, including support for user-initiated feedback to repair planning decisions or data produced by the QA components
- providing a mechanism to restart a planning session midway, to enable a user to run an alternate scenario or correct an error made by the system.

References

[1] J. Chu-Carroll, K. Czuba, J. Prager, and A. Ittycheriah. In question answering, two heads are better than one. In *Proceedings of HLT-NAACL 2003*, pages 24–31, 2003.

[2] A. Echihabi, U.Hermjakob, E. Hovy, D. Marcu, E. Melz, and D. Ravichandran. Multiple-engine question answering in textmap. In *Proceedings of the Twelfth Text REtrieval Conference (TREC2003)*, 2003.

[3] S. Harabagiu, D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Girju, V. Rus, and P. Moraresc u. The role of lexico-semantic feedback in open-domain textual question-answering. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics (ACL-2001)*, pages 274–281, July 2001.

[4] E. Nyberg, T. Mitamura, J. Callan, J. Carbonell, R. Frederking, K. Collins-Thompson, L. Hiyakumoto, Y. Huang, C. Huttenhower, S. Judy, J. Ko, A. Kupsc, L. V. Lita, V. Pedro, D. Svoboda, and B. Van Durme. The javelin question-answering system at trec 2003: A multi-strategy approach with dynamic planning. In *Proceedings of the Twelfth Text REtrieval Conference (TREC2003)*, 2003.

A The JAVELIN domain specification: `QA.domain`

```
(define (domain QA)

        (:types entity - qtype
                causation - qtype
                activity - qtype
                procedural - qtype
                vocabulary - qtype
                meaning - qtype
                biographic - qtype
                relationship - qtype
                temporal - atype
                location - atype
                numeric-expression - atype
                regexp - atype
                person-bio - description
                definition - description
                description - atype
                object - atype
                lexicon - atype
                person-name - proper-name
                organization-name - proper-name
                proper-name - atype
                relation - atype
                causal-antecedent - atype
                causal-consequence - atype
                process - atype
                action - atype
                qtype
                atype
                context
                aformat
                constraint
                atypename
                question
                docset
                fillset
                anslist
                ix-name
        )

        (:constants
                dt knn fst svm dict light - ix-name
                new - context
                ranked set known-qty - aformat
                temporal_t location_t numeric-expression_t - atypename
                person-bio_t definition_t lexicon_t description_t - atypename
                person-name_t organization-name_t proper-name_t - atypename
                regexp_t object_t relation_t process_t action_t - atypename
                causal-antecedent_t causal-consequence_t atype_t - atypename
        )

        (:predicates
                (interactive_session)
                (satisfies ?q - question ?a - atype ?al - anslist)
                (expected_ans_format ?f - aformat ?i - int)
                (request ?q - question ?r - qtype)
                (retrieved_docs ?d - docset ?r - qtype)
                (no_docs_found ?r - qtype)
                (no_more_docs ?r - qtype)
                (candidate_fills ?f - fillset ?r - qtype ?d - docset ?ix - ix-name)
                (no_fills_found ?r - qtype ?d - docset ?ix - ix-name)
                (ranked_answers ?a - anslist ?r - qtype ?f - fillset)
                (no_answers ?r - qtype ?f - fillset)
                (displayed ?a - anslist)
```

```
        (asked_about_atype ?r - qtype)
        (asked_about_keywords ?r - qtype)
        (server_down ?ix - ix-name)
)

(:metrics
        system_time
        request_quality
        docset_quality
        fillset_quality
        answer_quality
)

(:features
        (question_id ?q - qtype) - question
        (request_context ?q - qtype) - context
        (superlative_value ?q - qtype) - constraint
        (evaluation_value ?q - qtype) - constraint
        (quantity_value ?q - qtype) - int
        (ordinal_value ?q - qtype) - constraint
        (expected_atype ?q - qtype) - atypename
        (atype_confidence ?q - qtype) - float
        (qtype_confidence ?q - qtype) - float
        (extracted_terms ?q - qtype) - int
        (docset_size ?d - docset) - int
        (min_docs_requested ?d - docset) - int
        (max_docs_requested ?d - docset) - int
        (max_doc_score ?d - docset) - float
        (min_doc_score ?d - docset) - float
        (ave_doc_score ?d - docset) - float
        (fillset_size ?f - fillset) - int
        (max_fill_score ?f - fillset) - float
        (min_fill_score ?f - fillset) - float
        (ave_fill_score ?f - fillset) - float
        (alist_size ?a - anslist) - int
        (min_ans_score ?a - anslist) - float
        (max_ans_score ?a - anslist) - float
        (ave_ans_score ?a - anslist) - float
)

(:domain-functions
        (genReqObjID) - qtype
        (genDocsetID) - docset
        (genFillsetID) - fillset
        (genAnslistID) - anslist
        (genAnsID) - atype
        (estTimeRS ?a - atypename) - float
        (estTimeIX ?a - atypename ?ix - ix-name) - float
        (estTimeAG ?a - atypename) - float
        (estTimeResponse ?f - aformat) - float
        (probNoDocs ?q - qtype) - float
        (probNoMoreDocs ?q - qtype) - float
        (probDocsHaveAns ?q - qtype) - float
        (probDocsNoAns ?q - qtype) - float
        (probServerDown ?ix - ix-name) - float
        (probNoFills ?a - atypename ?d - docset) - float
        (probGoodFills ?a - atypename ?d - docset ?ix - ix-name) - float
        (probBadFills ?a - atypename ?d - docset ?ix - ix-name) - float
        (probNoAns ?q - qtype ?f - fillset) - float
        (probGoodAns ?q - qtype ?f - fillset) - float
        (probBadAns ?q - qtype ?f - fillset) - float
        (probAcceptAns ?q - qtype ?a - anslist) - float
        (probRejectAns ?q - qtype ?a - anslist) - float
        (estRequestQual ?r - question) - float
        (estDocsetQual ?q - qtype) - float
        (estFillsetQual ?q - qtype ?d - docset ?ix - ix-name) - float
        (estAnslistQual ?q - qtype ?f - aformat ?c - fillset) - float
```

```
)


(:action RETRIEVE_DOCUMENTS
        :param (?q - question ?ro - qtype)
        :precond (and (request ?q ?ro)
                  (not (no_docs_found ?ro))
                  (not (exists (?d - docset) (retrieved_docs ?d ?ro)))
                  (> (extracted_terms ?ro) 0)
                  (> request_quality 0))

        :dbind (?docs    (genDocsetID)
                ?dur      (estTimeRS (expected_atype ?ro))
                ?pnodocs (probNoDocs ?ro)
                ?pgood   (probDocsHaveAns ?ro)
                ?pbad    (probDocsNoAns ?ro)
                ?dqual   (estDocsetQual ?ro))

        :peffect
           (?pnodocs ((no_docs_found ?ro)
                      (scale-down request_quality 2)
                      (assign docset_quality 0)
                      (increase system_time ?dur))
            ?pgood   ((retrieved_docs ?docs ?ro)
                      (assign docset_quality ?dqual)
                      (increase system_time ?dur))
            ?pbad    ((retrieved_docs ?docs ?ro)
                      (scale-down request_quality 2)
                      (assign docset_quality 0)
                      (increase system_time ?dur)))

        :execute (RetrievalStrategist ?docs ?ro 10 15 300))


(:action EXTRACT_KNN_CANDIDATE_FILLS
        :param (?ro - qtype ?docs - docset)
        :precond (and (retrieved_docs ?docs ?ro)
                      (not (exists (?f - fillset)
                           (candidate_fills ?f ?ro ?docs knn)))
                      (not (no_fills_found ?ro ?docs knn))
                      (not (server_down knn))
                      (> docset_quality 0.3))

        :dbind (?fills    (genFillsetID)
                ?dur       (estTimeIX (expected_atype ?ro) knn)
                ?pdown     (probServerDown knn)
                ?pnofills (probNoFills (expected_atype ?ro) ?docs)
                ?pgood     (probGoodFills (expected_atype ?ro) ?docs knn)
                ?pbad      (probBadFills (expected_atype ?ro) ?docs knn)
                ?fqual     (estFillsetQual (expected_atype ?ro) ?docs knn))

        :peffect
           (?pdown    ((server_down knn)
                       (assign fillset_quality 0)
                       (increase system_time ?dur))
            ?pnofills ((no_fills_found ?ro ?docs knn)
                       (scale-down request_quality 2)
                       (scale-down docset_quality 2)
                       (assign fillset_quality 0)
                       (increase system_time ?dur))
            ?pgood    ((candidate_fills ?fills ?ro ?docs knn)
                       (assign fillset_quality ?fqual)
                       (increase system_time ?dur))
            ?pbad     ((candidate_fills ?fills ?ro ?docs knn)
                       (scale-down request_quality 2)
                       (scale-down docset_quality 2)
                       (assign fillset_quality 0)
```

```
                       (increase system_time ?dur)))

        :execute (KNNRequestFiller ?fills ?ro ?docs 300))


(:action EXTRACT_FST_CANDIDATE_FILLS
        :param (?ro - qtype ?docs - docset)
        :precond (and (retrieved_docs ?docs ?ro)
                      (not (exists (?f - fillset)
                           (candidate_fills ?f ?ro ?docs fst)))
                      (not (no_fills_found ?ro ?docs fst))
                      (not (server_down fst))
                      (> docset_quality 0.3))

        :dbind (?fills    (genFillsetID)
                ?dur      (estTimeIX (expected_atype ?ro) fst)
                ?pdown    (probServerDown fst)
                ?pnofills (probNoFills (expected_atype ?ro) ?docs)
                ?pgood    (probGoodFills (expected_atype ?ro) ?docs fst)
                ?pbad     (probBadFills (expected_atype ?ro) ?docs fst)
                ?fqual    (estFillsetQual (expected_atype ?ro) ?docs fst))

        :peffect
           (?pdown    ((server_down fst)
                       (assign fillset_quality 0)
                       (increase system_time ?dur))
            ?pnofills ((no_fills_found ?ro ?docs fst)
                       (scale-down request_quality 2)
                       (scale-down docset_quality 2)
                       (assign fillset_quality 0)
                       (increase system_time ?dur))
            ?pgood    ((candidate_fills ?fills ?ro ?docs fst)
                       (assign fillset_quality ?fqual)
                       (increase system_time ?dur))
            ?pbad     ((candidate_fills ?fills ?ro ?docs fst)
                       (scale-down request_quality 2)
                       (scale-down docset_quality 2)
                       (assign fillset_quality 0)
                       (increase system_time ?dur)))

        :execute (FSTRequestFiller ?fills ?ro ?docs 300))


(:action EXTRACT_SVM_CANDIDATE_FILLS
        :param (?ro - qtype ?docs - docset)
        :precond (and (retrieved_docs ?docs ?ro)
                      (not (exists (?f - fillset)
                           (candidate_fills ?f ?ro ?docs svm)))
                      (not (no_fills_found ?ro ?docs svm))
                      (not (server_down svm))
                      (> docset_quality 0.3))

        :dbind (?fills    (genFillsetID)
                ?dur      (estTimeIX (expected_atype ?ro) svm)
                ?pdown    (probServerDown svm)
                ?pnofills (probNoFills (expected_atype ?ro) ?docs)
                ?pgood    (probGoodFills (expected_atype ?ro) ?docs svm)
                ?pbad     (probBadFills (expected_atype ?ro) ?docs svm)
                ?fqual    (estFillsetQual (expected_atype ?ro) ?docs svm))

        :peffect
           (?pdown    ((server_down svm)
                       (assign fillset_quality 0)
                       (increase system_time ?dur))
            ?pnofills ((no_fills_found ?ro ?docs svm)
                       (scale-down request_quality 2)
                       (scale-down docset_quality 2)
```

```
                   (assign fillset_quality 0)
                   (increase system_time ?dur))
         ?pgood   ((candidate_fills ?fills ?ro ?docs svm)
                   (assign fillset_quality ?fqual)
                   (increase system_time ?dur))
         ?pbad    ((candidate_fills ?fills ?ro ?docs svm)
                   (scale-down request_quality 2)
                   (scale-down docset_quality 2)
                   (assign fillset_quality 0)
                   (increase system_time ?dur)))

     :execute (SVMRequestFiller ?fills ?ro ?docs 300))


(:action EXTRACT_LIGHT_CANDIDATE_FILLS
     :param (?ro - qtype ?docs - docset)
     :precond (and (retrieved_docs ?docs ?ro)
                  (not (exists (?f - fillset)
                       (candidate_fills ?f ?ro ?docs light)))
                  (not (no_fills_found ?ro ?docs light))
                  (not (server_down light))
                  (> docset_quality 0.3))

     :dbind (?fills    (genFillsetID)
             ?dur      (estTimeIX (expected_atype ?ro) light)
             ?pdown    (probServerDown light)
             ?pnofills (probNoFills (expected_atype ?ro) ?docs)
             ?pgood    (probGoodFills (expected_atype ?ro) ?docs light)
             ?pbad     (probBadFills (expected_atype ?ro) ?docs light)
             ?fqual    (estFillsetQual (expected_atype ?ro) ?docs light))

     :peffect
        (?pdown    ((server_down light)
                   (assign fillset_quality 0)
                   (increase system_time ?dur))
         ?pnofills ((no_fills_found ?ro ?docs light)
                   (scale-down request_quality 2)
                   (scale-down docset_quality 2)
                   (assign fillset_quality 0)
                   (increase system_time ?dur))
         ?pgood   ((candidate_fills ?fills ?ro ?docs light)
                   (assign fillset_quality ?fqual)
                   (increase system_time ?dur))
         ?pbad    ((candidate_fills ?fills ?ro ?docs light)
                   (scale-down request_quality 2)
                   (scale-down docset_quality 2)
                   (assign fillset_quality 0)
                   (increase system_time ?dur)))

     :execute (LIGHTRequestFiller ?fills ?ro ?docs 300))

(:action RANK_CANDIDATES
     :param (?ro - qtype ?docs - docset ?fills - fillset
             ?form - aformat ?ix - ix-name ?i - int)
     :precond (and (candidate_fills ?fills ?ro ?docs ?ix)
                  (expected_ans_format ?form ?i)
                  (not (no_answers ?ro ?fills))
                  (not (exists (?a - anslist)
                               (ranked_answers ?a ?ro ?fills)))
                   (> fillset_quality 0))

     :dbind (?ans   (genAnslistID)
             ?dur   (estTimeAG (expected_atype ?ro))
             ?pnone (probNoAns ?ro ?fills)
             ?pgood (probGoodAns ?ro ?fills)
             ?pbad  (probBadAns ?ro ?fills)
             ?aqual (estAnslistQual ?ro ?fills ?form))
```

```
        :peffect
           (?pnone ((assign answer_quality 0)
                    (assign fillset_quality 0)
                    (no_answers ?ro ?fills)
                    (increase system_time ?dur))
            ?pgood ((assign answer_quality ?aqual)
                    (ranked_answers ?ans ?ro ?fills)
                    (increase system_time ?dur))
            ?pbad  ((scale-down docset_quality 2)
                    (scale-down fillset_quality 2)
                    (assign answer_quality 0)
                    (ranked_answers ?ans ?ro ?fills)
                    (increase system_time ?dur)))

        :execute (AnswerGenerator ?ans ?ro ?fills ?ix 300))

(:action CHECK_ANSWERS
        :param (?q - question ?ro - qtype ?fills - fillset
                ?ans - anslist ?form - aformat ?i - int)
        :precond (and (not (interactive_session))
                      (request ?q ?ro)
                      (ranked_answers ?ans ?ro ?fills)
                      (not (displayed ?ans))
                      (expected_ans_format ?form ?i)
                      (> answer_quality 0))

        :dbind (?a     (genAnsID)
                ?dur   (estTimeResponse ?form))

        :peffect
           (1.0 ((satisfies ?q ?a ?ans)
                 (displayed ?ans)
                 (assign answer_quality 1)
                 (increase system_time ?dur)))

        :execute (CheckAnswers ?a ?ans ?q ?form))

(:action RESPOND_TO_USER
        :param (?q - question ?ro - qtype ?fills - fillset
                ?ans - anslist ?form - aformat ?i - int)
        :precond (and (interactive_session)
                      (request ?q ?ro)
                      (ranked_answers ?ans ?ro ?fills)
                      (not (displayed ?ans))
                      (expected_ans_format ?form ?i)
                      (> answer_quality 0))

        :dbind (?a     (genAnsID)
                ?dur   (estTimeResponse ?form)
                ?pgood (probAcceptAns ?ro ?ans)
                ?pbad  (probRejectAns ?ro ?ans))

        :peffect
           (?pgood ((satisfies ?q ?a ?ans)
                    (displayed ?ans)
                    (assign answer_quality 1)
                    (increase system_time ?dur))
            ?pbad  ((displayed ?ans)
                    (scale-down request_quality 2)
                    (scale-down docset_quality 2)
                    (scale-down fillset_quality 2)
                    (assign answer_quality 0)
                    (increase system_time ?dur)))

        :execute (RespondToUser ?a ?ans ?q ?form))
```

```
(:action ASK_USER_FOR_ANSWER_TYPE
        :param (?q - question ?ro - qtype ?docs - docset
                ?fills - fillset ?ans - anslist ?form - aformat)
        :precond (and (interactive_session)
                      (request ?q ?ro)
                      (not (asked_about_atype ?ro))
                      (or (and (ranked_answers ?ans ?ro ?fills)
                               (< (max_ans_score ?ans) 0.1))
                          (no_docs_found ?ro)
                          (exists (?ix - ix-name)
                                  (no_fills_found ?ro ?docs ?ix))
                          (exists (?f - fillset ?x - ix-name)
                                  (and (candidate_fills ?f ?ro ?docs ?x)
                                  (no_answers ?ro ?f)))))
        :dbind (?ro2   (genReqObjID)
                ?dur   (estTimeResponse ?form))
        :peffect
           (0.2  ((increase system_time ?dur)
                  (request ?q ?ro2)
                  (asked_about_atype ?ro))
            0.8  ((increase system_time ?dur)
                  (asked_about_atype ?ro)))

        :execute (AskUserForAtype ?q ?ro ?ro2))


(:action ASK_USER_FOR_MORE_KEYWORDS
        :param (?q - question ?ro - qtype ?docs - docset
                ?fills - fillset ?ans - anslist ?form - aformat)
        :precond (and (interactive_session)
                      (request ?q ?ro)
                      (not (asked_about_keywords ?ro))
                      (or (and (ranked_answers ?ans ?ro ?fills)
                               (< (max_ans_score ?ans) 0.1))
                          (no_docs_found ?ro)
                          (exists (?ix - ix-name)
                                  (no_fills_found ?ro ?docs ?ix))
                          (exists (?f - fillset ?x - ix-name)
                                  (and (candidate_fills ?f ?ro ?docs ?x)
                                  (no_answers ?ro ?f)))))

        :dbind (?ro2   (genReqObjID)
                ?dur   (estTimeResponse ?form))
        :peffect
           (0.1  ((increase system_time ?dur)
                  (asked_about_keywords ?ro))
            0.9  ((increase system_time ?dur)
                  (request ?q ?ro2)
                  (asked_about_keywords ?ro)))

        :execute (AskUserForKeywords ?q ?ro ?ro2)))
```

B GUI-Planner DTDs and Field Descriptions

B.1 Question XML sent by the GUI

```
<!ELEMENT   ANSWERQUESTION   (#PCDATA)>
<!ATTLIST   ANSWERQUESTION
            type             (new | continuation)   #IMPLIED
            interactive      (true | false)         #IMPLIED
            log              CDATA                  #IMPLIED
            collection       CDATA                  #IMPLIED
            amount           CDATA                  #IMPLIED
            atype            CDATA                  #IMPLIED
            trecID           CDATA                  #IMPLIED
            utility-thresh   CDATA                  #IMPLIED
            success-thresh   CDATA                  #IMPLIED
            time             CDATA                  #IMPLIED>
```

Three of the optional attributes are provided for test purposes in conjunction with the AnswerOracle: the ‘trecID’ attribute can be used to pass the TREC id of the question to the Oracle; the ‘amount’ and ‘atype’ attributes are used to correct list and definition question classifications produced by the question analysis (by modifying the corresponding RequestObject produced by the QuestionAnalyzer).

Attribute	**Possible Values**	**Description**
type	new | continuation	Specifies whether the question context is new or a continuation of the previous request.
interactive	true | false	Indicates whether the planner is allowed to request feedback from the user during question processing.
log	*<IPaddress>*:*<port>* *e.g.,* 128.2111.11:1111	Specifies a host and port to which the planner should send log messages during processing. If omitted, no log messages are sent.
collection	*any of the predefined RS document collection names, e.g.* TREC, AQUAINT, CNS, DICT	Specifies the collection to search.
utility-thresh	*float between 0 and 1*	Sets the planner goal utility threshold.
success-thresh	*float between 0 and 1*	Sets the planner success-likelihood threshold.
time	*positive integer*	Sets a time limit (in seconds) for the question-answering process.
amount	multiple	Specifies the number of answers being sought. Currently, the Planner recognizes the value ‘multiple’ as indicating the question is a list question.
atype	*any of the predefined QA answer-type categories, e.g.,* definition	Specifies the expected answer type of the question.
trecID	L?[1-9][0-9]*	Provides the TREC ID of the question (used in tests with the AnswerOracle).

B.2 Answer XML returned by the Planner

```
<!ELEMENT   ANSWERLIST   (ANSWER*)>
<!ATTLIST   ANSWERLIST
            question_id  CDATA        #REQUIRED>

<!ELEMENT   ANSWER       (#PCDATA)>
<!ATTLIST   ANSWER
            id           CDATA        #REQUIRED
            confidence   CDATA        #REQUIRED>
```

Element/Attribute	Possible Values	Description
question_id	*positive integer*	Unique repository identifier for the question being answered.
ANSWER	*text*	An answer string extracted by the QA system.
id	*positive integer*	Unique repository identifier for the answer.
confidence	*float between 0 and 1*	Confidence score assigned to the answer.

B.3 Dialog XML sent by the Planner

```
<!ELEMENT   DIALOG     (QUESTION, CHOICE*)>
<!ATTLIST   DIALOG
            type       (yes/no | multiple-choice | text)   #REQUIRED
            default    CDATA                               #IMPLIED>

<!ELEMENT   QUESTION   (#PCDATA)>
<!ELEMENT   CHOICE     (#PCDATA)>
```

Element/Attribute	Possible Values	Description
type	yes/no | multiple-choice | text	Declares the type of dialog the GUI should display.
default	*text*	Used with yes/no or multiple-choice dialogs to specify a default response.
QUESTION	*text*	The question to pose to the user.
CHOICE	*text*	Used with multiple-choice dialogs; defines one of the choices to display.

B.4 Load XML sent by the GUI

```
<!ELEMENT   DOMAINFILE    (#PCDATA)>
<!ELEMENT   PROBLEMFILE   (#PCDATA)>
```

The value of each element is a unix-style filename, either with a full path specification or a path relative to the default domain directory (i.e., the value of DomainDirDefault in the configuration file).

C Planner-EM DTDs and Field Descriptions

C.1 Session ID XML returned by the EM

```
<!ELEMENT   PlannerID   EMPTY>
<!ATTLIST   PlannerID
            id          CDATA    #REQUIRED>
```

The PlannerID id attribute is a positive integer specifying the unique repository identifier assigned to the current planning session.

C.2 Execution XML sent by the Planner

```
<!ELEMENT   Execute      (Command+)>
<!ATTLIST   Execute
            version      CDATA            #REQUIRED
            exe_id       CDATA            #REQUIRED
            session_id   CDATA            #REQUIRED>

<!ELEMENT   Command      (Assigns?, Arg*)>
<!ATTLIST   Command
            name         CDATA            #REQUIRED>

<!ELEMENT   Assigns      (#PCDATA)>
<!ATTLIST   Assigns
            object       CDATA            #REQUIRED>

<!ELEMENT   Arg          (#PCDATA)>
<!ATTLIST   Arg
            name         CDATA            #REQUIRED>
```

<table>
<tr><th>Element/Attribute</th><th>Possible Values</th><th>Description</th></tr>
<tr><td>version</td><td>positive float</td><td>XML DTD version id.</td></tr>
<tr><td>exe_id</td><td>positive integer</td><td>Planner-generated id for this execution request.</td></tr>
<tr><td>session_id</td><td>positive integer</td><td>Repository id for this planning session. In combination with the exe_id, this uniquely identifies the execution request in the repository.</td></tr>
<tr><td>Command name</td><td>QuestionAnalyzer | RetrievalStrategist | (KNN|FST|SVM|LIGHT)RequestFiller | AnswerGenerator</td><td>Name of the QA module to call.</td></tr>
<tr><td>object</td><td>RequestObject | DocumentSet | RequestFillSet | AnswerList</td><td>Type of object the Planner expects the execution call to produce.</td></tr>
<tr><td>Assigns</td><td>[A-Z]+[0-9]+</td><td>Planner-generated id to assign to the object being produced by the execution call (e.g., ‘DS4475’ in ‘<Assigns object=”DocumentSet”>DS4475</Assigns>’)</td></tr>
<tr><td>Arg</td><td>(see table below)</td><td>Module-specific input value.</td></tr>
<tr><td>Arg name</td><td>(see table below)</td><td>Name of the module-specific input.</td></tr>
</table>

The following table defines the module arguments currently in use for the JAVELIN system. Because these arguments are subject to relatively frequent changes as new QA components become available and existing components are revised, all module-specific argument name/value pairs are determined by the operator execution specifications in the planning domain file loaded at run-time rather than defined as full-fledged elements in the DTD. This has the advantage of obviating the need to recompile the Planner Module executable when these input arguments change; revisions can be incorporated just by updating domain file used by the planner (and making any necessary updates to the EM).

Arg name	**Supplied To**	**Req'd?**	**Possible Arg Values**	**Description**
Question	QA	Y	*text*	Question text sent by the GUI.
RequestObject	RS,IX,AG	Y	RO[0-9]+	Planner id of the RequestObject to supply as input.
Collection	RS	N	*any of the predefined RS document collection names*	Specifies the collection to search.
Mindoc	RS	N	*positive integer*	Minimum number of documents the RS should return.
Maxdoc	RS	N	*positive integer*	Maximum number of documents the RS should return.
Filter	RS	N	*positive integer specifying an internal document id*	Specifies a document to filter out from the results set.
DocumentSet	IX	Y	DS[0-9]+	Planner-generated id of the DocumentSet to extract candidates from.
RequestFillSet	AG	Y	FS[0-9]+	Planner-generated id of the RequestFillSet to select an answer from.
IXType	AG	Y	FST | KNN | LIGHT | SVM	Specifies which version of the IX module generated the candidates.
Time	*all*	N	*positive integer*	Timeout limit (secs) for a response.

C.3 Results XML returned by the EM

```
<!ELEMENT   Results      (#PCDATA)>
<!ATTLIST   Results
            version      CDATA      #REQUIRED
            exe_id       CDATA      #REQUIRED
            session_id   CDATA      #REQUIRED
            EM_time      CDATA      #REQUIRED>
```

Attribute	**Possible Values**	**Description**
version	*positive float*	XML DTD version id.
exe_id	*positive integer*	Planner-generated id for this execution request.
session_id	*positive integer*	Repository id for this planning session.
EM_time	*positive float*	EM's self-estimated processing time (in seconds) taken to service the execution request.

The contents of the ‘Results’ element is the XML produced by the module that was executed.

C.4 Object modification request XML sent by the Planner

```
<!ELEMENT   ObjectModification   (ObjectToUpdate+)>
<!ATTLIST   ObjectModification
            version              CDATA                        #REQUIRED
            session_id           CDATA                        #REQUIRED>

<!ELEMENT   ObjectToUpdate       (Replace*, Remove*, Add*, Require*)>
<!ATTLIST   ObjectToUpdate
            type                 CDATA                        #REQUIRED
            id                   CDATA                        #REQUIRED
            newid                CDATA                        #REQUIRED>

<!ELEMENT   Replace              (QuestionType | AnswerType | Document | RequestFill | Answer)>

<!ELEMENT   Remove               (Keyword | Document | RequestFill | Answer)>

<!ELEMENT   Add                  (Keyword)>

<!ELEMENT   Require              (Keyword)>

<!ELEMENT   QuestionType         (#PCDATA)>
<!ATTLIST   QuestionType
            confidence           CDATA                        #REQUIRED>

<!ELEMENT   AnswerType           (#PCDATA)>
<!ATTLIST   AnswerType
            confidence           CDATA                        #REQUIRED>

<!ELEMENT   Keyword              (#PCDATA)>
<!ATTLIST   Keyword
            type                 (word | phrase | proper)     #REQUIRED>

<!ELEMENT   Document             EMPTY>
<!ATTLIST   Document
            trecID               CDATA                        #REQUIRED
            confidence           CDATA                        #REQUIRED>

<!ELEMENT   RequestFill          EMPTY>
<!ATTLIST   RequestFill
            id                   CDATA                        #REQUIRED
            confidence           CDATA                        #REQUIRED>

<!ELEMENT   Answer               EMPTY>
<!ATTLIST   Answer
            rank                 CDATA                        #REQUIRED
            confidence           CDATA                        #REQUIRED>
```

Element/Attribute	Possible Values	Description
ObjectModification version	*positive float*	XML DTD version id.
ObjectModification session_id	*positive integer*	Repository id for this planning session.
ObjectToUpdate type	RequestObject | DocumentSet | RequestFillSet | AnswerList	Type of object to update.
ObjectToUpdate id	[A-Z]+[0-9]+	Planner id associated with object being updated.
ObjectToUpdate newid	[A-Z]+[0-9]+	New planner id to associate with the cloned and updated copy of the object
QuestionType	*any of the predefined QA question-type categories, e.g.,* entity	New question-type to assign.
QuestionType confidence	*float between 0 and 1*	Confidence in the accuracy of the new question type.
AnswerType	*any of the predefined QA answer-type categories, e.g.,* location	New answer-type to assign.
AnswerType confidence	*float between 0 and 1*	Confidence in the accuracy of the new answer type.
Keyword	*text*	Term or phrase related to the question.
Keyword type	(word | phrase | proper)	Type of the keyword.
Document trecID	*external document id, e.g.,* NYT19981215.0157	Unique id of a document (in the TREC and AQUAINT collections, the id within the DOCNO tag of the document).
Document confidence	*float between 0 and 1*	Score to assign to the document indicating its relevance likelihood.
RequestFill id	*positive integer*	List order in which the candidate was returned by the IX (i.e., the first candidate listed in the IX output is given an id of '1').
RequestFill confidence	*float between 0 and 1*	Score to assign to the candidate indicating the likelihood it is an answer.
Answer rank	*positive integer*	Rank order for the answer in the AG output.
Answer confidence	*float between 0 and 1*	Score to assign to the answer indicating the likelihood it is correct.

C.5 Planner data XML to be stored in the repository

```
<!ELEMENT   InitialState        (Action, BeliefState, Goal, UtilityFunction)>
<!ATTLIST   InitialState
            version             CDATA                       #REQUIRED
            question_id         CDATA                       #REQUIRED
            session_id          CDATA                       #REQUIRED>
<!ELEMENT   Action              (#PCDATA)>
<!ATTLIST   Action
            id                  CDATA                       #REQUIRED>
<!ELEMENT   BeliefState         (State+)>
<!ATTLIST   BeliefState
            id                  CDATA                       #REQUIRED>
<!ELEMENT   State               (MetricSet?, Objects?, Literals?)>
<!ATTLIST   State
            id                  CDATA                       #REQUIRED
            prob                CDATA                       #REQUIRED
            util                CDATA                       #REQUIRED>
<!ELEMENT   MetricSet           (Metric+)>
<!ELEMENT   Metric              EMPTY>
<!ATTLIST   Metric
            name                CDATA                       #REQUIRED
            value               CDATA                       #REQUIRED>
<!ELEMENT   Objects             (#PCDATA)>
<!ELEMENT   Literals            (#PCDATA>
<!ELEMENT   Goal                (#PCDATA)>
<!ATTLIST   Goal
            Gthresh             CDATA                       #REQUIRED
            Sthresh             CDATA                       #REQUIRED>
<!ELEMENT   UtilityFunction     (Function+)>
<!ELEMENT   Function            EMPTY>
<!ATTLIST   Function
            name                CDATA                       #REQUIRED
            param               CDATA                       #REQUIRED
            weight              CDATA                       #REQUIRED>

<!ELEMENT   ExecutionOutcome    (Action, BeliefState)>
<!ATTLIST   ExecutionOutcome
            version             CDATA                       #REQUIRED
            exe_id              CDATA                       #REQUIRED
            applied_to          CDATA                       #REQUIRED
            session_id          CDATA                       #REQUIRED>
<!ELEMENT   PlanningStep        (CandidateAction*, Outcome)>
<!ATTLIST   PlanningStep
            version             CDATA                       #REQUIRED
            session_id          CDATA                       #REQUIRED>
<!ELEMENT   CandidateAction     (Action, BeliefState)>
<!ATTLIST   CandidateAction
            applicable_in       CDATA                       #REQUIRED
            EU                  CDATA                       #REQUIRED>
<!ELEMENT   Outcome             EMPTY>
<!ATTLIST   Outcome
            status              (OK | NoCandidates)         #REQUIRED
            addedToPlan         CDATA                       #REQUIRED>
```

Element/Attribute	Possible Values	Description
InitialState version	*positive float*	XML DTD version id.
InitialState question_id	*positive integer*	Repository id for the current question.
InitialState session_id	*positive integer*	Repository id for this planning session.
Action	*text*	Planner-generated text description consisting of the planning operator's name and execution specification.
Action id	A[0-9]+	Planner-generated id for the action; unique within a planning session.
BeliefState id	B[0-9]+	Planner-generated id for a planning belief state.
State id	S[0-9]+	Planner-generated information state id.
State prob	*float between 0 and 1*	Likelihood of being in the state.
State util	*float between 0 and 1*	Estimated utility of being in the state.
Metric name	[A-Z_]+	Name of a planning metric defined in the planning domain.
Metric value	*float*	Associated value of the planning metric.
Objects	*text*	Comma separated list of planning state objects (as *name:type pairs).*
Literals	*text*	Comma separated list of planning state literals.
Goal	*text*	Literal condition the planner must satisfy to achieve the goal.
Goal Gthresh	*float between 0 and 1*	Planner goal utility threshold.
Goal Sthresh	*float between 0 and 1*	Planner success-likelihood threshold.
Function name	[A-Za-z_]+	Name of a planning utility function defined in the planning domain.
Function param	[A-Z_]+	Name of the planning metric used as the argument to the function.
Function weight	*float between 0 and 1*	Relative weight assigned to this function in the overall calculation of the utility.
ExecutionOutcome version	*positive float*	XML DTD version id.
ExecutionOutcome exe_id	*positive integer*	Planner-generated id for this execution request.
ExecutionOutcome session_id	*positive integer*	Repository id for this planning session.
ExecutionOutcome applied_to	B[0-9]+	Planner-generated id of the planning belief state in which the action was executed.
PlanningStep version	*positive float*	XML DTD version id.
PlanningStep session_id	*positive integer*	Repository id for this planning session.
CandidateAction applicable_in	B[0-9]+	Planner-generated id for the planning belief state in which the candidate action applies.
CandidateAction EU	*float between 0 and 1*	Expected utility of executing the candidate action in the belief state.
Outcome status	(OK | NoCandidates)	Indicates whether the planner was able to extend the plan or failed to extend the plan (because no candidate actions were available).
Outcome addedToPlan	A[0-9]+	Planner-generated id of the action selected to extend the plan.

C.6 Batch request XML sent by the Planner

```
<!ELEMENT   BatchRequest   (Command)>
<!ATTLIST   BatchRequest
            version        CDATA            #REQUIRED>

<!ELEMENT   Command        (Assigns?, Arg*)>
<!ATTLIST   Command
            name           CDATA            #REQUIRED>

<!ELEMENT   Assigns        (#PCDATA)>
<!ATTLIST   Assigns
            object         CDATA            #REQUIRED>

<!ELEMENT   Arg            (#PCDATA)>
<!ATTLIST   Arg
            name           CDATA            #REQUIRED>
```

Element/Attribute	**Possible Values**	**Description**
BatchRequest version	*positive float*	XML DTD version id.
Command name	Initialize | Terminate | StartQuestion | EndQuestion	Type of batch test request.
Arg	*(see table below)*	Command-specific input value.
Arg name	*(see table below)*	Name of the command-specific input.

Arg name	**Used With**	**Req'd?**	**Possible Arg Values**	**Description**
TrecID	StartQuestion	Y	L?[1-9][0-9]*	The TREC ID of the question.
MRR	Terminate	Y	*positive float between 0 and 1*	Average mean-reciprocal rank score for the batch test.
TrecScore	Terminate	Y	*positive float between 0 and 1*	Average accuracy score for the batch test.
QuestionFile	Terminate	Y	*text*	Name of the file containing the list of questions tested.
DomainFile	Terminate	Y	*text*	Name of the QA domain file the planner used during the test.
Description	Terminate	Y	*text*	Description of the batch test; used as a label on the JAVELIN test results web page.

C.7 Batch data XML returned by the EM

```
<!ELEMENT   BatchData     (BatchDir|CachedXMLs)>
<!ATTLIST   BatchData
            version       CDATA                  #REQUIRED>

<!ELEMENT   BatchDir      (#PCDATA)>

<!ELEMENT   CachedXMLs    (File*)>

<!ELEMENT   File          (#PCDATA)>
```

<table>
<tr><th>Element/Attribute</th><th>Possible Values</th><th>Description</th></tr>
<tr><td>BatchData version</td><td>positive float</td><td>XML DTD version id.</td></tr>
<tr><td>BatchDir</td><td>unix directory (full path)</td><td>Name of the directory where the EM will write the cached files, e.g., ‘/usr0/htdocs/javelin/Planner/0402_601’</td></tr>
<tr><td>File</td><td>[A-Z]+_(In|Out)put_qL?[0-9]+_ro[0-9]+.txt</td><td>Cached input or output file name consisting of an uppercase module identifier (e.g., ‘QA’ or ‘IXF’), input/output designation, the TREC id of the question, and its corresponding RequestObject id.</td></tr>
</table>