

Introspective Predicates for Explaining Task Execution in CALO

Alyssa Glass
Artificial Intelligence Center
SRI International

Deborah McGuinness
Knowledge Systems, Artificial Intelligence
Laboratory (KSL)
Stanford University

Introduction

The CALO system will include an end-to-end system capability that will allow CALO to provide explanations to users in response to questions about its execution and reasoning. This document focuses on explanations for task-oriented processing within the CALO system. We plan to support a wide range of task explanations, providing CALO users with the ability to ask detailed questions about task execution and to engage in a back-and-forth dialogue with CALO about its past, current, and future task execution. To provide these detailed explanations of the behavior of the CALO Task Manager, execution traces must be annotated with enough meta-data to support a wide range of behaviors. Particularly when answering complex questions, an explanation system must have access to information about many aspects of execution and planning.

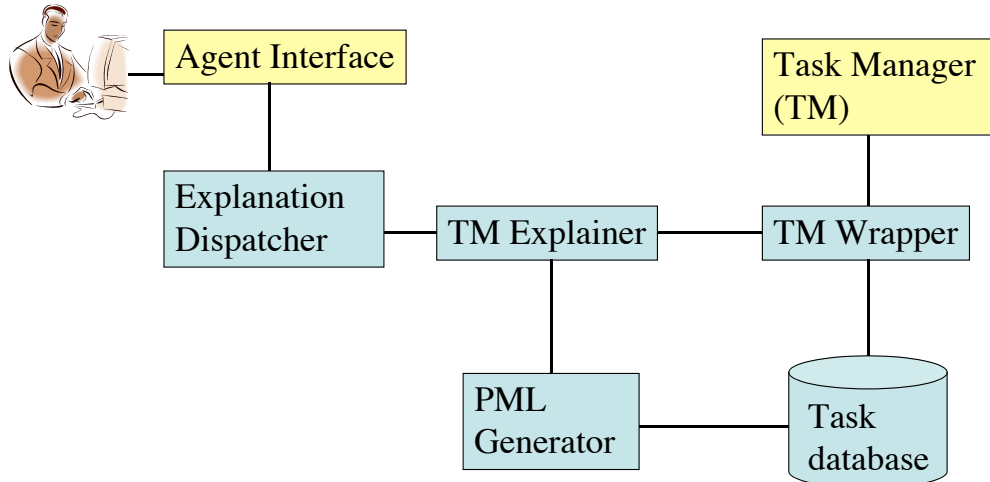
In this document, we outline the needs that should be met by the Task Manager to allow usable, detailed explanations to be generated. These needs are organized around categories of "introspective predicates" – queryable facts that the Task Manager should support to provide necessary information to the explanation system.

Architecture

To motivate the discussion in this document, we first provide an overview of the architecture of our explanation system. The main components of this system are shown below. User questions about task processing are received through the Task Manager graphical interface and passed to the Explanation Dispatcher. The Explanation Dispatcher can accept a broad range of task-related questions as outlined in the document "Plan for Explaining Task Execution in CALO." The Explanation Dispatcher forwards these task-related questions to the Task Manager (TM) Explainer, which determines the types of information needed for a response, and then requests the necessary information from the Task Manager through the Task Manager Wrapper. This Wrapper dumps the information into a database, where it is used to generate PML to support the explanation. This PML proof is then used as the basis for a variety of possible answers and follow-up questions, augmented as necessary with additional queries to the Task Manager Wrapper.

An initial prototype version of this system, using simple PML generated from limited data gathered by the SPARK Wrapper, has been implemented and was demonstrated in the CALO Year 2 Video Demonstration system. We now plan to support a wider range of task explanations by gathering much more detailed information through the Task Manager wrapper, further expanding the

information that can be stored in the database for use in generating PML justifications. The additional introspection predicates described below are motivated by CALO's coverage of the range of queries described in the task execution explanation paper. Current information supports only a subset of the simpler questions and limits follow-up question options.



Motivating Scenarios

To motivate the predicates discussed below, we first present two scenarios that will be used to test task explanations in the CALO system.

The first scenario presented here is from the reimbursement domain. For this scenario, we assume the following general process for receiving reimbursement for an expense. Note that this is a simplification of the process used by the CALO Task Manager, but is sufficient to illustrate the explanations discussed below. For CALO to receive reimbursement for its user, it follows four basic steps. First, CALO fills out a reimbursement form, collecting and organizing information that it knows about its user and the item requiring reimbursement, seeking additional information from outside sources as well as direct questions to the user as necessary. Second, CALO presents the completed reimbursement form to its user for approval. Third, CALO submits the approved form to an administrative assistant who reviews the form and forwards it to the appropriate finance organization so that a reimbursement payment can be made. At this point in the process, we note that there is “wait” condition imposed on CALO from an outside agent; CALO cannot proceed with the final step in the process until the external processing has been completed and a check has arrived. Once this external process completes, CALO completes step four, notifying the user that the reimbursement funds have arrived.

A summary of this reimbursement process:

1. Fill out reimbursement form
 - a. Identify information needs
 - b. Gather information from multiple sources

2. Present reimbursement form to user for approval
 - a. Notify user of needed action
 - b. Monitor user and remind user as needed
3. Submit reimbursement form for processing
→ WAIT for reimbursement check ...
4. Notify user that reimbursement check has arrived

The second scenario is from the equipment purchase domain. This scenario has been the focus testbed for the task learning work within CALO; we include it here in order to firmly ground our work on explaining learned and modified tasks. In this scenario, CALO follows three basic steps in order to purchase new equipment for the user. The first task is to get price quotes for the desired equipment. The process requires three quotes from three different sources. The second task is to get approval for the purchase from two authorizing agents. This task is completed when a particular form is signed by the approval organization representatives. The final task is to submit the purchase order. Before this task can start, the system needs to know that the correct form was signed. This task is terminated when a particular requisition form has been sent to purchasing. Note that this scenario, like the previous one, is essentially sequential in nature, and has a step which requires the system to wait for outside agents to perform their own tasks before the system can continue.

A summary of this equipment purchase scenario:

1. Get Purchase Quotes
 - a. Gather requirements from the user
 - b. Collect quotes from multiple on-line sources
 - c. Fill out purchase order form
2. Get Purchase Approval
 - a. Contact appropriate approval authorities
 - b. Monitor responses
3. Submit Purchase Order

Introspective Predicates

Below we discuss each of the introspective predicates identified as necessary for an explainable task execution system. Since we are primarily motivated by explaining the SPARK-based CALO Task Manager, where necessary we include notes which help to map these predicates into the SPARK representations necessary to gather the information. We also note where these alignments are non-trivial or warrant further discussion.

For each predicate, we also indicate a strategy and sample question and explanation response that is enabled by the availability of that predicate, as described in the document "Plan for Explaining Task Execution in CALO" and taken from the reimbursement domain described above. These strategies and explanations are meant only as examples; for many of these predicates, multiple explanations in response to multiple questions would utilize the provided predicate information.

The predicates are divided into three sections, described below.

Basic Procedure Information

The predicate information listed here is not generally dependent on when or how an action begins. It could be preloaded into a database, updated only when new procedures are added or existing procedures are modified. Or, this information could be provided using persistent predicates in SPARK that are only off-loaded to the DB when relevant to the procedures being reported.

1. AUTHOR of a procedure

This predicate indicates the initial source of the knowledge contained in a procedure; not who requested its execution. The author could be a person (who wrote this procedure), or it could be a system (for automatically generated procedures). Procedures that are written/created by one author and then modified by someone else are annotated with modification predicates as indicated below.

Explanation enabled: Identify source

Example:

Why are you waiting for approval?

I am waiting for approval because my learning module told me that getting approval is a necessary subtask of getting reimbursement.

2. SIGNIFICANCE FLAGS indicating display information about each procedure

Significance flags are used to filter out low-level tasks, such as system control or internal meta-procedures that are generally not relevant to a user's understanding of a procedure. This flag could have multiple levels, with an explanation drilling down to reveal more levels as a user asks more complex questions about a particular task. Currently, SPARK has some primitive ways of identifying which sub-tasks are useful to initially display for a user and which ones are not, but this capability needs to be formally defined and expanded.

Explanation enabled: Provide an abstraction of task outline

Example:

What are you doing right now?

I am gathering information to fill out a reimbursement form.

(Explanation does not drill down to describe the exact information-gathering procedures.)

3. Information about LEARNED and MODIFIED procedures

Rather than a single predicate, procedure learning covers a set of predicates that can be used together to indicate how a procedure has been created or modified. These predicates are tailored towards procedures that are learned through automated learning systems, but can be used for any procedure whose definition changes over time. Included in this set of predicates:

- a. What algorithm and/or program was used to learn the procedure. We draw a distinction here between the AUTHOR of a procedure as described above, and the algorithm used by that AUTHOR to do the specific procedure creation or modification indicated here.
- b. Date and time indicating when this version of the procedure was added
- c. Information about the data used to do the learning. Initially this may only contain sources used but could later be expanded to include more granularity as usage dictates.
- d. A comment from the user, to be used in cases where the user directly instructed an automated system on how to create/modify a procedure. This comment field could also be used by the author of hand-written procedures.
- e. Which lines of the procedure are different, if this can be indicated in cases where an existing procedure has been modified.
- f. A link back to the original procedure as it appeared before this modification, or to similar procedures upon which a new procedure has been based.
- g. Date/time span over which learning occurred in the case of learned procedures. For instance, in the case of learning by observation, the observation period could be an extended period of time before the insertion of the new procedure.

This set of predicates, or an appropriate subset if necessary, would be stored each time a new procedure is added or an existing procedure is modified. Thus, if a particular procedure is updated twice, complete information about each of these modifications would be available, enabling the system to understand the evolution of the procedure over time. Note that these predicates require SPARK to maintain outdated procedures that are no longer considered for execution.

Explanation enabled: Show task modification

Example:

Why did you email Sally before submitting the purchase order?

This step was added on August 3, when you instructed me, "email my assistant before submitting purchase orders."

In addition to enabling several types of explanations, information about modified procedures also enables new classes of follow-up questions. For instance, tracing procedure modifications over time allows the system to compare different ways of performing tasks:

Explanation enabled: Follow-up after providing an abstraction of the task plan

Example:

Why didn't you get a second purchase approval this time?

On June 23, you instructed me, "you do not need a second approval when the cost is less than \$1500." The cost of this purchase was \$1257.34, so this new condition was met."

Explanation enabled: Expose task learning

Example:

How did you learn how to fill out a purchase order form this way?

I learned how to fill out a purchase order form by learning by observation on September 25-26.

Execution Information

This information is generated as a procedure begins being executed, and remains valid in some form throughout the execution of that task. This information also includes history related to completed tasks. Since the system will be generating a lot of data, storing it all internally is not a scalable long-term solution. We expect to use a model where the data is updated dynamically in SPARK, and then off-loaded to a DB either as a stream or in a batch. A batch of data could be off-loaded when a question is asked of the system. Alternatively, whenever a top-level task completes (either successfully or in a failure) all history information related to that task and its subtasks could be off-loaded to the DB.

4. REQUESTOR of a procedure

The requestor is the person or agent who initiated the execution of a procedure. Initially, the requestor of a procedure will generally be the user. Some tasks can be proactively initiated by CALO itself. In future versions of the system, tasks can also be assigned by external agents. The requestor will be stored in the same format as the author above.

Explanation enabled: Identify requestor

Example:

Why are you getting reimbursement?

Sally told me to.

5. TERMINATION requirements

CALO needs a method of representing termination requirements for currently executing tasks. In SPARK, it is not always clear how termination requirements are represented. In cases where there are explicit termination conditions in a SPARK procedure, we expect to be able to formally represent these conditions (and any known bindings) using PPDR [PHM+05]. For other termination conditions (e.g., the completion of subtasks) we will need to decide on a principled way of describing the required information concerning termination requirements.

Explanation enabled: Identify termination conditions

Example:

Why are you submitting a reimbursement form?

I am trying to get reimbursement, and I can only do it by submitting a reimbursement form for approval.

6. Task START TIME

The actual time when each task and subtask begins execution, represented in iCal format.

Explanation enabled: Expose temporal information

Example:

When did you submit the reimbursement form?

I submitted the reimbursement form on Tuesday at 4:15pm.

7. Task END TIME

The actual time when each task and subtask completed (either successfully or through failure), represented in iCal format.

Explanation enabled: Expose temporal information

Example:

When did you finish filling out the reimbursement form?

I finished filling out the reimbursement form at 2:15pm.

8. Task DEADLINE

If a deadline is being imposed on a task and/or subtask, it is represented here, in iCal format.

Explanation enabled: Expose deadlines

Example:

When is the deadline for submitting the reimbursement?

I must submit the reimbursement by 5:00pm.

9. Task EXPECTED DURATION

Expected durations can be stored both for currently executing tasks and for future planned tasks. For future tasks, expected durations are static, but once a task begins executing, the expected duration can be updated based on conditions at the time. Thus, we include expected duration here, with execution predicates.

Explanation enabled: Provide a time estimate

Example:

When will I receive my reimbursement?

Reimbursements take 5 business days when requested at the end of the month, so I expect your reimbursement to arrive next Tuesday.

10. Task STATUS

Possible status indicators: in process, waiting, failed, completed successfully, past deadline, desired. We are soliciting information about any other status indicator requests.

Explanation enabled: Provide execution details

Example:

Are you having trouble completing the reimbursement submission?

Filling out the reimbursement form and getting approval completed successfully, and submit for reimbursement is in process.

11. WAIT CONDITIONS (will need full binding information)

For any task currently holding due to a “wait” condition, we will need to store complete information about the conditions, including expected duration of the wait. These conditions might be seen, or at least represented, as a subcategory of the termination conditions described above.

Explanation enabled: Provide progress details

Example:

What are you doing?

I am waiting for Joe to approve the reimbursement. I have been waiting for 2 days.

Expected wait time is 5 days.

12. Previous completed SUBTASKS of the current goal

For any task (either currently executed or already completed) we should be able to reconstruct the complete sequence of subtasks that have executed or are currently executing. When combined with the temporal information described above, this information will allow us to answer questions like, which subtask took the longest, which subtask seemed to have encountered problems during execution, etc.

Explanation enabled: Expose task structure

Example:

What have you done to get my reimbursement?

I have finished filling out the reimbursement form and submitting it for approval and signature.

13. Previous TOP-LEVEL GOALS completed before this one

We need to store all of the predicates listed in this document for past tasks as well as for tasks that are currently relevant. If necessary, we can timebox the information that we store, either by the number of tasks (e.g., keep the last 100 top levels goals and all relevant additional information) or by elapsed time (e.g., keep the history of everything we’ve done in the last month) or both.

Explanation enabled: Execution history

Example:

What have you done this morning?

I finished scheduling a meeting. I finished buying a laptop. I started planning your business trip to San Diego.

14. Tasks that MISSED DEADLINES

As we keep track of deadline information, this predicate will be added to flag any procedures that miss their deadline, regardless of any action that is taken to remedy the situation. Having this information easily available will enable us to quickly answer questions about when something started to go wrong with a procedure, as well as provide insight into when the overall system might have been overloaded with tasks.

Explanation enabled: Expose temporal information

Example:

Are you behind schedule on any tasks?

I need to submit reimbursement form by Thursday at 1:00, but it is now Thursday at 1:30 and I have not yet finished the previous task of receiving approval.

15. Expose PRECONDITIONS

While the static preconditions for a given procedure could be included in the basic procedure information described above, we instead include precondition predicates here so that they may be stored with applicable variable bindings that will only be known during execution. This information will allow CALO to:

- a. Identify which preconditions were met, and with which bindings, when a procedure was begun.
- b. Identify preconditions that were tested and NOT met, with which bindings, and what action was taken (try a different procedure; fail; wait for conditions to be met later, ...) when a procedure was attempted.
- c. When asked, identify which preconditions are valid in the current state and with what bindings, thus allowing answers to questions about which procedures COULD be started in the present state.

Explanation enabled: Expose preconditions

Example:

Why didn't you get reimbursement from petty cash?

Getting reimbursement from petty cash requires the reimbursement amount to be less than \$50. This reimbursement was for \$70.

16. FAILURE REASON

As we begin to explain failures in the Task Manager, we will need to keep track of what went wrong, as a first step towards explaining WHY something went wrong. We will need to identify what types of failures we can encounter in SPARK (that is worthy of explaining), how we want to categorize them, and what information will be needed to support failure explanation. For now, complete failure explanation is beyond the scope of this document, so we include this predicate only as a placeholder for more detailed failure predicates. As a start, we suggest a simple categorization of why a deadline might have been missed (e.g., because the system was busy; because an outside agent failed to perform a subtask; because a system subtask failed without sufficient time to recover). We are also exploring Leake's categorization scheme [Lea92] for anomalies.

Projection (Future) Information

This information is not normally generated by the system, but can be generated on demand as necessary to answer a question. It includes both questions about task execution that has not yet happened, and questions about alternatives at decision points that have already passed.

17. Task EXPECTED START TIME (for future planned tasks or subtasks)

Information about expected start time should be generated, on demand, both for future subtasks of a current goal, as well as for desired top-level goals that have not yet begun execution.

Explanation enabled: Expose temporal projections

Example:

When will you begin submitting the reimbursement form for processing?
I expect to begin submitting the form for processing at 5:00 today.

18. Expected/planned NEXT SUBTASK

Ideally, we would like to be able to answer arbitrary questions about future task execution. For now, we limit ourselves to simply asking about the next subtask planned for a currently goal. We anticipate that, as SPARK develops more forward-looking capabilities, we will be able to provide these projections to the user, as well as provide information about the confidence level in these projections, complete expected start times for all projections, and why we believe these tasks will be executed.

Explanation enabled: Expose execution projections

Example:

What are you doing next to get reimbursement?
After I finish my current task of identifying information needed for the reimbursement form, I will begin gathering information for the form.

19. Expose ALL REMAINING SUBTASKS

As noted above, this type of projection is not currently available in SPARK, except in very limited situations. For example, for tasks that have no conditional branching, we will be able to reveal at least the tasks that we know to be necessary, given the current execution goal.

Explanation enabled: Expose execution projections

Example:

What will you do to get reimbursement?
I am currently filling out the reimbursement form. I still need to get approval for the reimbursement form, submit the form for reimbursement, wait for the reimbursement, and notify you when the reimbursement is ready.

20. Identify ALTERNATIVE PROCEDURES that also satisfy a given goal

Given a particular goal, identify a set of procedures that satisfy that goal. This information would not only allow explanations of why a particular procedure was chosen, but would also allow more open-ended questions (e.g., If I asked you to do X, how would you do it?). There remains an open question about how these procedures should be identified. Current possibilities include reference by name, by procedure steps, by preconditions (and/or termination conditions).

Explanation enabled: Expose alternative procedures

Example:

Why did you get reimbursement through the finance department?

I also know how to get reimbursement from petty cash, but the necessary preconditions for that procedure were not met.

Status

We have begun to implement the introspective predicates above that are well defined. We have extended SPARK to include support for these predicates. We are working with Wolverton to help him use these predicates to expand the SPARK database wrapper work so that it may provide expanded database dumps including the output from these predicates. Further, we are using the Inference Web [McP04] PML generation services to generate PML from that database and then using Inference Web browsing services to display the proofs. Implementation is in prototype stage. It has been integrated into the CALO year 2 video and was demonstrated at the Year 3 kickoff meeting. We have begun collaboration with Blythe (ISI) to explain the results of learning by instruction. In this context, we are also testing our set of predicates in a learning setting to evaluate their adequacy.

References

[Lea92] David Leake, *Evaluating Explanations: A Content Theory*. Hillsdale, NJ: Lawrence Erlbaum Associates. 1992.

[McP04] Deborah L. McGuinness and Paulo Pinheiro da Silva. **Explaining Answers from the Semantic Web: The Inference Web Approach**. *Journal of Web Semantics*. Vol.1 No.4., pages 397-413, October 2004.

[PHM+05] Paulo Pinheiro da Silva, Patrick Hayes, Deborah L. McGuinness, Richard Fikes and Priyendra Deshwal. **Towards Checking Hybrid Proofs**. *Technical Report KSL-05-01*, Knowledge Systems Laboratory, Stanford University, USA, 2005.