2008 IEEE International Conference on
Robotics and Automation
Pasadena, CA, USA, May 19-23, 2008

A Deliberative Architecture for AUV Control

Conor McGann, Frederic Py, Kanna Rajan, Hans Thomas, Richard Henthorn, Rob McEwen
Monterey Bay Aquarium Research Institute, Moss Landing, California
{cmcgann,fpy,kanna.rajan,hthomas,henthorn,rob}@mbari.org

Abstract—**Autonomous Underwater Vehicles (AUVs) are an increasingly important tool for oceanographic research demonstrating their capabilities to sample the water column in depths far beyond what humans are capable of visiting, and doing so routinely and cost-effectively. However, control of these platforms to date has relied on fixed sequences for execution of pre-planned actions limiting their effectiveness for measuring dynamic and episodic ocean phenomenon. In this paper we present an agent architecture developed to overcome this limitation through on-board planning using Constraint-based Reasoning. Preliminary versions of the architecture have been integrated and tested in simulation and at sea.**

I. Introduction

Oceanography has traditionally relied on ship-based observations. These have recently been augmented by robotic platforms such as Autonomous Underwater Vehicles (AUV) [1-3], which are untethered powered mobile robots able to carry a range of payloads efficiently over large distances in the deep ocean. A common design relies on a modular tube-like structure with propulsion at the stern and various sensors, computers and batteries taking up the bulk of the tube (Fig. 1). AUVs have demonstrated their utility in oceanographic research in gathering time series data by repeated water-column surveys [4], detailed bathymetric maps of the ocean floor in areas of tectonic activity [5,6] and performed hazardous under-ice missions [7].

Typically AUVs do not communicate with the support ship or shore while submerged and rely on limited stored battery packs while operating continuously for tens of hours. Current AUV control systems [8] are a variant of the behavior-based Subsumption architecture [9]. A behavior is a modular encapsulation of a specific control task and includes acquisition of a GPS fix, descent to a target depth, drive to a given waypoint, enforcement of a mission depth envelope etc. An operator defines each plan as a collection of behaviors with specific start and end times as well as maximum durations, which are scripted a priori using simple mission planning tools. In practice, missions predominantly consist of sequential behaviors with duration and task specific parameters equivalent to a linear plan with limited flexibility in task duration. Such an approach becomes less effective as mission uncertainty increases. Further, the architecture offers no support to manage the potentially complex interactions that may result amongst behaviors, pushing a greater cognitive burden on behavior developers and mission planners. This paper describes an automated onboard planning system to generate robust mission plans using system state and desired goals. By capturing explicit interactions between behaviors as plan constraints in the domain model and the use of goal-oriented commanding, we expect this approach to reduce the cognitive burden on AUV operators. Our interest in the near term is to incorporate decision-making capability to deal with a range of dynamic and episodic ocean phenomenon that cannot be observed with scripted plans.

Fig 1. An MBARI AUV at sea

The remainder of this paper is laid out as follows. Section II lays out the architecture of our autonomy system, section III details the experimental results to date, related work follows in section IV with concluding remarks in section V.

II. The T-REX Architecture

T-REX (Teleo-Reactive EXecutive) is a goal-oriented system, with embedded automated planning [14,15] and adaptive execution. It encapsulates the long-standing notion of a *sense-deliberate-act* cycle in what is typically considered a hybrid architecture where sensing, planning and execution are interleaved. In order to make embedded planning scalable the system enables the scope of deliberation to be partitioned functionally and temporally and to ensure the current state of the agent is kept consistent and complete during execution. While T-REX was built for a specific underwater robotics application, the principles behind its design are applicable in any domain where deliberation and execution are intertwined.

Fig. 2 shows a conceptual view of a Teleo-Reactive *Agent*. An agent is viewed as the *coordinator* of a set of concurrent control loops. Each control loop is embodied in a Teleo-Reactor (or reactor for short) that encapsulates all

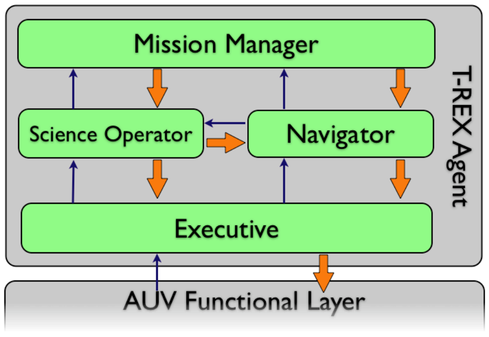

Fig. 2: A 4-reactor T-REX agent.

details of how to accomplish its control objectives. Arrows represent a messaging protocol for exchanging facts and goals between reactors: thin arrows represent *observations* of current state; thick arrows represent *goals* to be accomplished. Reactors are differentiated in 3 ways:

- Functional scope: indicating the state variables of concern for deliberation and action.
- Temporal scope: indicating the look-ahead window over which to deliberate.
- Timing requirements: the latency within which this component must deliberate for goals in its planning horizon.

Fig. 2 for example, shows four different reactors; the *Mission Manager* provides high-level directives to satisfy the scientific and operational goals of the mission: its temporal scope is the whole mission, taking minutes to deliberate if necessary. The *Navigator* and *Science Operator* manage the execution of sub-goals generated by the *Mission Manager*. The temporal scope for both is in the order of a minute even as they differ in their functional scope. Each refines high-level directives into executable commands depending on current system state. The *Science Operator* is able to provide local directives to the *Navigator*. For example if it detects an ocean front it can request the navigation mode to switch from a Yo-Yo pattern in the vertical plane to a Zig-Zag pattern in the horizontal plane, to have better coverage of the area. Deliberation may safely occur at a latency of 1 second for these reactors. The *Executive* provides an interface to a modified version of the existing AUV functional layer. It encapsulates access to commands and vehicle state variables. The *Executive* is reasonably approximated as having zero latency within the timing model of our application since it will accomplish a goal received with no measurable delay, or not at all; in other words it does not deliberate.

T-REX has a central and explicit notion of time with all reactors synchronized by an internal clock. The unit of time is a *tick*, defined in external units on a per application basis; tick boundaries signify when synchronization of all reactors must occur while between ticks reactors may deliberate. The agent-state is represented as a set of timelines, which capture the evolution of a system state-variable over time. A timeline is a sequence of *tokens* that are temporally qualified assertions expressed as a predicate with start and end time bounds defining the temporal scope over which it holds. The minimum duration of a token is a tick giving a discrete synchronous view of the state of the world. Token start and end times can be defined as intervals to express temporal flexibility.

Agent timelines are distributed across reactors depending on their functional scope. Information exchange between reactors, where necessary, is provided through the following mechanisms:

- Explicit timeline ownership: Each timeline is owned by exactly one reactor. Any reactor may request a new goal, or replan such requests in the event of a change of plan; but only the owner of the timeline can decide what goal to instantiate.

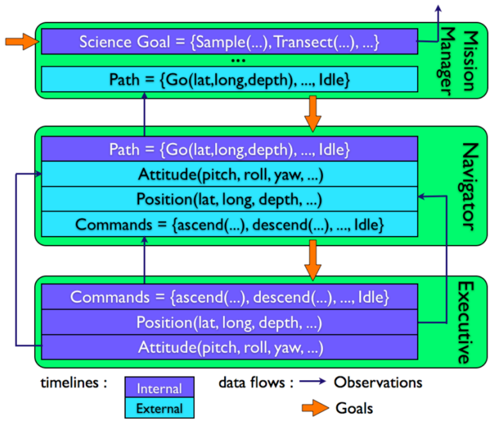
Science Goal = {Sample(...), Transect(...), ...}
...
Path = {Go(lat,long,depth), ..., Idle}
Mission Manager
Path = {Go(lat,long,depth), ..., Idle}
Attitude(pitch, roll, yaw, ...)
Position(lat, long, depth, ...)
Commands = {ascend(...), descend(...), ..., Idle}
Navigator
Commands = {ascend(...), descend(...), ..., Idle}
Position(lat, long, depth, ...)
Attitude(pitch, roll, yaw, ...)
Executive
timelines :
Internal
External
data flows :
Observations
Goals

Fig 3: Information flow between timelines

- Observations: capture the current value of a timeline. Observations are asserted by the owner of a timeline.
- Goals: express a desired future timeline value. They offer a way to delegate a task to a reactor. Goals are requested for expansion into sub-goals or commands and can be recalled on plan changes when replanning is triggered.
- Dispatch and notification rules: define when information must be shared to ensure consistency and completeness of agent state at the execution frontier and to allow sufficient time for deliberation.

The mapping between reactors and timelines is the basis for sharing information. If a reactor owns a timeline it is declared *internal* to that reactor; if it uses a timeline to observe values and/or express requirements it is declared *external* to that reactor. Fig. 3 illustrates the flow of information in a system containing 3 reactors: The *Mission Manager* keeps track of science goals to give directives to the *Navigator* using the *Path* external timeline. The *Navigator* manages the navigation of the AUV with one *internal* timeline and three external timelines. The navigation route is used to select the appropriate commands to send to the *Executive* as an internal timeline while Position and Attitude timelines capture AUV navigation data. A Command timeline captures the command state of the *Executive*. These external timelines are *internal* to the *Executive* in turn. The Command timeline values are the actual commands that are managed by the AUV functional layer. The content of this last timeline at the execution frontier corresponds to the currently active behavior.

To ensure a complete and consistent view of system state, the T-REX information exchange framework needs to impose further restrictions on the way timelines, observations and goals can be used:

- No 'holes' are allowed at the execution frontier i.e. all timelines must have a value at the end of the current tick.
- If no update is provided via an observation, and in the absence of information to the contrary, a reactor assumes the previous value(s) on the timeline is/are still valid. We

refer to this as the Inertial Value Assumption since it conveys some inherent inertia of current values. Contradictory information can come from the model or from a new observation. This has important implications for reducing the cost of synchronization since observations need only be published as timeline values change.

- At the end of the current tick, all observations must be consistent, by requiring all reactors to hold the same view at the execution frontier.
- The past is monotonic. All tokens that have finished or that have started but have yet to finish (i.e. they span the execution frontier) can only be restricted in time.
- An observation received at a tick applies to that tick only. It cannot refer to the past except by restricting the values of a token that is actually running (i.e. with an end time in the future). It cannot refer to the future, as it would then be a goal, rather than observed reality.

```
handleTick(tick){
  synchronize (tick);
  dispatchGoals(tick);
  done = false;
  while(!done && currentTick() == tick)
     done = stepNextReactor(tick);}
```

Fig. 4: The T-REX agent algorithm

The algorithm at the heart of a T-REX agent in Fig. 4 is called at the start of every tick. There are three key steps in the algorithm; first, all timelines are synchronized at the current execution frontier which is followed by the dispatch of goals. And finally, the remaining CPU time can be allocated to reactors for deliberation in incremental steps. Each of these component algorithms operates over the entire set of reactors.

A. *Synchronization*

The goal of synchronization is to produce a consistent and complete view of agent state at the execution frontier. All reactors synchronize at the same rate – once per tick. While this may seem onerous, the actual cost of synchronization is based on how much information has actually changed. For example, in Fig. 3. the Position timeline is relatively volatile and will likely change on every tick. However, the Path timeline may hold a single value for many ticks. In this case, as a result of the *Inertial Value Assumption*, if no new observation is received, the Path timeline will extend its current value simply by incrementing the lower bound of the end time of the current value.

The strict rules of timeline ownership enable a clear policy for conflict resolution: observations dominate expectations. For example, if the *Navigator* expected the vehicle depth to be less than 0.3m in order in order to obtain a GPS fix but the actual depth observed by the *Executive* is 1 meter, then the expected value is discarded. This may impact plan feasibility and force the *Navigator* to find an alternative solution by rejecting the current plan.

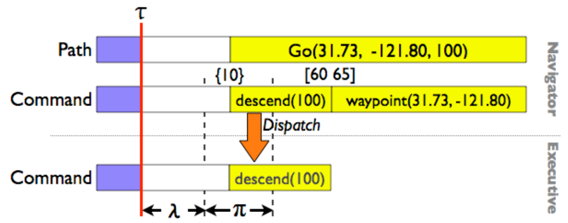

Fig. 5: Illustration of goal dispatching window

To ensure global consistency the agent undertakes local synchronization of the reactors until quiescence. In principle, this operation is equivalent to solving a planning problem over the set of all *internal* timelines for a planning horizon restricted to a tick. If a reactor has an external timeline, it depends on its owner for such consistency. In this way the reactors form a dependency graph which in practice we require to be acyclic, allowing ordering of synchronization for purposes of efficiency.

B. *Dispatching Goals*

Where observations are the driver for reaction, goals are the driver for deliberation. The purpose of dispatching is to task reactors with new goals in a timely manner. To accomplish this, T-REX provides explicit parameters and rules to govern dispatching.

- λ - The latency of the reactor or the worst-case number of ticks to deliberate over a request.
- π - The planning horizon of the reactor quantifying the look-ahead for deliberation.
- τ - The execution frontier expressing the boundary between the past and the future.

To understand the implications of the above parameters, consider the example given in Fig. 5. To satisfy the goal *Go*(31.73, -121.80, 100) in its *Path* timeline the *Navigator* decides that it needs the vehicle to *descend*(100) at tick 10 for a duration between 50 and 55 ticks and then to achieve *waypoint*(31.73,-121.80) on successful termination of *descend.* Since the *Executive* is the owner of the *Command* timeline*,* these two goals need to be dispatched by the *Navigator* to the *Executive* so that the latter can resolve them. The importance of λ is to ensure the *Executive* has sufficient time to complete deliberation prior to starting the requested goal. If the start-time for a goal dispatched to the *Executive* at τ were necessarily less than $\tau+\lambda_{Exec}$ the *Executive* may be unable to deliberate to resolve the goal, leading to a plan failure.

Since the planning window of the *Executive* is π_{Exec}, the Executive should receive all goals that can start before $\tau+\lambda_{Exec}+\pi_{Exec}$. This will enable the *Executive* to leverage as much information as it can handle in making judicious decisions on how to accomplish the goals requested. Sending a goal with a start time *strictly greater* than $\tau+\lambda_{Exec}+\pi_{Exec}$ will not be considered by the *Executive*. Moreover, such dispatch incurs a cost for transmission of information and may over-commit the *Navigator* unnecessarily.

Therefore the general rule is that the *dispatching window for a timeline is a time window that depends on the latency and the look ahead of the reactor owning the timeline*. This dispatch window, H_D is therefore defined by:

$$H_D = [\tau + \lambda, \tau + \lambda + \pi]$$

This implies that as soon as the start time of a goal on an external timeline intersects H_D, it is dispatched to the owner of the timeline. This rule is necessary and sufficient to ensure that each reactor has sufficient time (λ) and information (π) to deliberate on goals provided by other reactors. In our implementation, we have an *Executive*, which is purely reactive and therefore $\lambda_{Exec} = \pi_{Exec} = 0$ implying that the *Executive* does not plan beyond the execution frontier.

C. *Deliberation*

The framework presented thus far makes the details of deliberation an internal concern for each reactor even if it has to capture different functional and temporal scope. Our own implementation of T-REX uses a Constraint-based Temporal Planning approach based on EUROPA-2 [10,11] employing a declarative model-based paradigm. The model describes state variables (e.g. position, battery level) and actions (e.g. *ascend*, *descend*, *getGPS*, *takeWaterSample*) of the system. Constraints can be specified to enforce relationships between state variables. For example, it is convenient to represent the vehicle as being at the surface, or not, which can be captured with a boolean state variable (e.g *AtSurface*). We define a relationship between this variable and the deph of the vehicle as follows: if *depth* <= 0.3 then *AtSurface* = true. The model also describes constraints between states and actions. For example, the vehicle must be at the surface during *getGPS*. A sample domain model is shown in Fig. 6 with the Path timeline having two predicates *At* and *Go*; the example rules in the parameter specification express the constraint that to be at a location, the AUV needs to go to that coordinate and the position must be maintained for a temporal interval that is consistent with the rest of the model. A T-REX agent uses a single model for control at various levels of abstraction and at various speeds of execution. Different reactors reference subsets of this model according to their functional scope.

```
class Path extends AgentTimeline {
  predicate At{Node location;}
  predicate Go{Node from; Node To;}
}
class Position extends AgentTimeline {
   predicate Holds{Node value};
}
Path::At {
   met_by(Go g);
   eq(g.to, location);
   contained_by(Position.Holds p);
   eq(p.value, location);
}
Path::Go {
   met_by(At p);
   eq(p.location, from);}
```

Fig. 6: A domain model in T-REX

The Deliberative reactor is a specialization of a Teleo-Reactor utilizing models, plans and planning to accomplish reactive and goal directed control. Fig. 7 describes the main components of this reactor. The inward pointing arrows reflect the

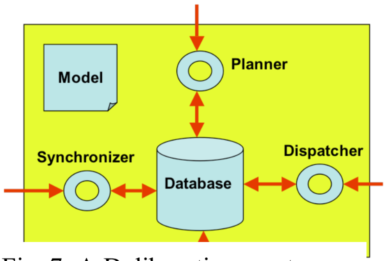

Fig. 7: A Deliberative reactor

invocations of the agent control loop for synchronization, dispatch and deliberation. The Database is a source and sink for observations and goals based on the semantics of internal and external timelines and the rules of information exchange. It is an extension of the EUROPA-2 plan database, augmented for specialized buffering for efficient access to timeline data for dispatch and synchronization and manages state information. Model rules are applied automatically through a combination of propositional inference and constraint propagation [21], to check consistency and prune infeasible elaborations of the plan maintained in the database. The Synchronizer is a specialized configuration of a EUROPA solver operating over a 1-tick horizon. It accomplishes local consistency and completeness. The database propagates the results of synchronization to the future. The Dispatcher is a simple algorithm that publishes goals to owner reactors of its external timelines according to the dispatch semantics previously defined. Finally, the Planner is yet another instance of a EUROPA solver used to deliberate over the specified temporal and functional scope of the reactor using a heuristic based chronological backtracking search for partial plan refinement. These entities together are used under different configurations for the Mission Manager, Science Operator and Navigator shown in the example in Fig 3. Further details on EUROPA can be found in [10,11].

III. EXPERIMENTAL RESULTS

Our experiments with T-REX at sea involved using two onboard computers on our AUV: a main vehicle computer a 244 Mhz PC/104 stack running the QNX real-time operating system running the functional layer, and a separate 367 MHz EPIC EPX-GX500 AMD Geode stack running Linux and T-REX. Communication between T-REX and the functional layer computer was with a socket-based protocol allowing the exchange of goals and state updates. For validation purposes we initially ran experiments on a high-fidelity AUV simulator based on [13] which captures vehicle dynamics to validate our missions. Sea trials with T-REX

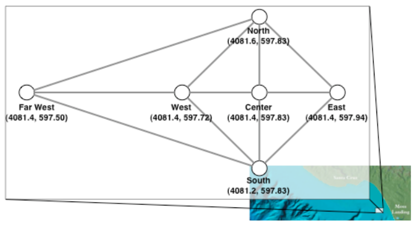

Fig. 8. A Mission traversal graph

onboard an AUV were in the Monterey Bay, California using our support ship the R/V Zephyr. The tick duration was set to 1 second. In this section we discuss one such mission among many executed at sea, where we focused on demonstrating nominal mission scenarios where scientists orient observations along specific legs.

One such set of legs was encoded as a graph located in the northern end of the Bay (Fig. 8). Such a representation has a number of distinct advantages; first it accurately predicts lower bounds on traversals from one node in the graph to another and thereby quantifies time and resources towards goal achievement (or for shedding over-subscribed goals). Second, it allows us to naturally deal with shortest path computations using our planner's existing constraint network algorithms and representation. Finally it allows scientists to clearly represent their requirements in a compact representation not unlike existing transect patterns with the important addition of specifying meta-level features such as goal priorities without concern for *how* the AUV would achieve these goals.

In Fig. 9, the goal of the mission was to head to the south node of such a traversal graph. The straight-line transect planned was repeatedly interrupted in-situ during deliberation, with check-in windows forcing the vehicle to surface every 100 seconds with at least 40 seconds at the surface. The dynamics of the vehicle [24] resulted in the vehicle to damp its downward decent by compensating on its pitch axis prior to a straight and level move thru the water column. This was soon followed by an *ascend* to the surface for a GPS fix followed by a short burst by the AUV to accelerate to depth. The mission goals are decomposed to those on the navigation timeline as a series of *Go*(South) followed by check-in tokens. Further decomposition of the *Go* activity in turn, results in *setpoint*, *descend* and *waypoint* tokens also within the *Navigator*. The *waypoint* token tries to achieve reaching the South node; however the 100sec check-in window constraint preempts the achievement of this traversal making the AUV surface.These series of actions are successively generated till an observation from the executive determines that the vehicle is indeed at the South node. Within each set of these *setpoint*, *descend* and *waypoint* tokens there is an important issue T-REX has to deal with in terms of execution uncertainty; in this case the precise end time for descending to depth is uncontrollable, i.e only exogenous conditions can determine the precise duration of this activity. The *waypoint* token duration therefore is limited by the durations of the *Go*, *setpoint* and *descend* tokens and can *only* be executed when the *descend* token finishes. Fig. 10 shows two examples where a *descend* could take longer (a) or shorter (b); modeled durations of the *descend* token however need to be able to reasonably encapsulate such variations which in practice are already considered when scripting plans a priori.

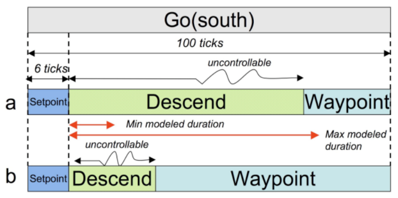

Fig. 10. A Token flexibility example

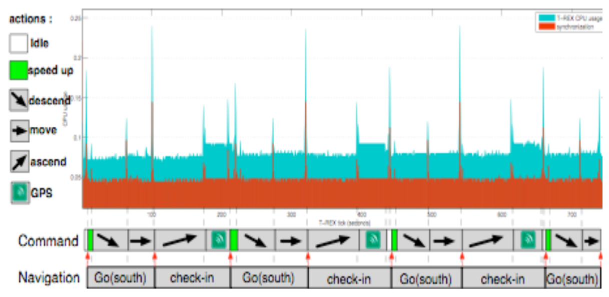

Fig. 11. CPU usage for the mission in Fig 9.

Fig. 11 shows the CPU usage and the impact of synchronization and deliberation that lead to changes in multiple reactors. The spikes shown correspond to a dynamic plan repair associated with the insertion of a check-in window. When the Executive terminates the waypoint activity, an observation is returned comparing the *Goto* location (South in metric units) with the (open-loop) distance traversed by the vehicle. If the vehicle is not at its desired *Goto* location, an additional *Goto* goal will be generated to make up the difference. The most common reason for waypoint terminating before reaching the target destination is due to duration constraints imposed by a check-in window. The Navigator inserts a check-in goal and

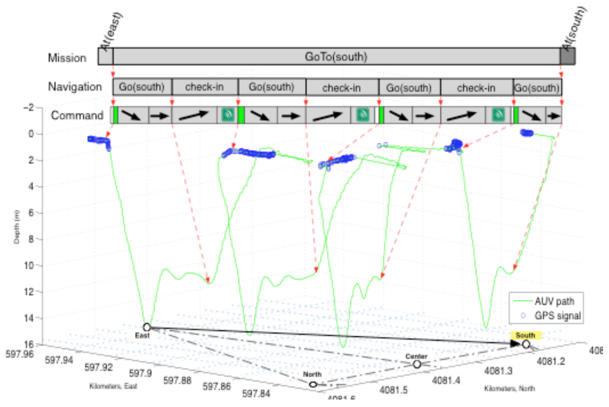

Fig. 9. A mission to traverse to the South node

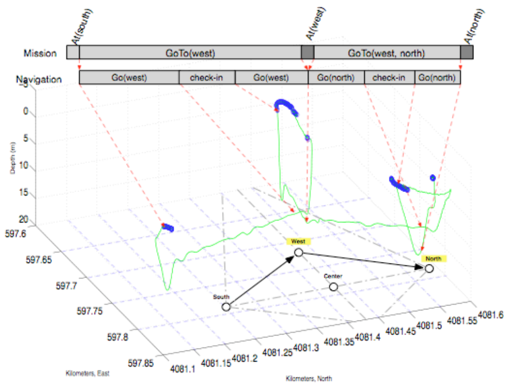

Fig. 12. T-REX plan with heading changes

further decomposes the goal in-situ as mentioned above into *ascend*, *getgps and setpoint* activities as needed. If on completion of a waypoint the vehicle is within an expected distance of its target location the Navigator will terminate the higher-level navigation goal. An interesting feature of the localization activity is the requirement that the AUV stay on the surface for at least 40 secs. However, as shown in the second check-in window in Fig. 11, when the vehicle is able to obtain a GPS fix well *under* this time limit the planner reactively inserts an *Idle* activity.

Fig. 12 illustrates a longer mission where T-REX received the goals to be *At* the West node and then *GoTo* the North node starting from the South node in the traversal graph. As before, we see the *Navigator* refining these goals with an interesting twist; when the AUV is at the West node, T-REX realizes that it had sufficient time to start the new goal before the next check-in and inserted a *Go(North)* token for the remaining duration. Such opportunistic decision-making is unrealizable with scripts designed a-priori and clearly demonstrates advantages of onboard deliberation. Additional data on T-REX test results can be found at [25].

IV. Related Work

T-REX is inspired from IDEA [16,17], which in turn is based on ideas in the Remote Agent Experiment (RAX) [14,15]. T-REX is similar to both in its formulation of a timeline-based representation, and in its use of planning and execution at its core. It is distinct from IDEA primarily in its formulation for exchanging and synchronizing state between reactors. The Autonomous Sciencecraft Experiment [18] conceptually borrows from RAX. The CASPER planner is not directly embedded in the execution as in T-REX. Further, temporal flexibility within and deals only with grounded plan representation. The 3-layered LAAS architecture [19] provides decisional capabilities using a constraint-based symbolic planner integrated with reactive components. However its disparate components are manipulating different representations using heterogeneous modeling languages. Such an approach tends to make system design and integration difficult [20]. In contrast, although T-REX's design leads to factoring of computation into layers, in practice a hierarchical structure is not inherent, nor is deliberation required or prohibited for any layer.

While a number of control architectures have been built for AUV control [1,8] T-REX's design philosophy is closest to DAMN [22] and ORCA [23]. DAMN is a reactive Subsumption based architecture with no inherent deliberation. ORCA uses schemas within a case-based planning framework; however the efficacy of ORCA's approach is unclear in terms of scalability in the number of schemas since the literature does not indicate whether the system was actually fielded on an AUV. Further there is no indication that it reasons explicitly with time and resources.

V. Conclusions and Future Work

Our results to date show that onboard planning and execution within the T-REX framework can handle uncertainty in the sub-sea domain gracefully well within the computational capacity available on our AUV's. Our immediate next steps are to integrate resource constraints for deliberation in goal selection and to demonstrate dynamic re-planning onboard the vehicle to adapt to science observations opportunistically to enable characterization of dynamic and episodic phenomenon such as ocean Fronts and Thin Layers.

VI. Acknowledgements

This research was supported by the David and Lucile Packard Foundation. We thank the crew of the R/V Zephyr for their help in deployments and NASA Ames Research Center for making the EUROPA planner available.

References

[1] Yuh, J., "Design and Control of Autonomous Underwater Robots: A Survey" Autonomous Robots, 2000 8, 7–24, 2000.
[2] Blidberg D.R., "Autonomous Underwater Vehicles: A Tool for the Ocean," Unmanned Systems, vol. 9, pp. 10-15, Spring 1991
[3] Bellingham J. G., et.al. "A Second Generation Survey AUV" In IEEE Conf on Autonomous Underwater Vehicles, Cambridge, MA, 1994.
[4] Ryan, J.P., et.al "Physical-biological coupling in Monterey Bay, California: topographic influences on phytoplankton", Marine Ecology Progress Series, Vol 287: 28-32 2005.
[5] Yoerger, D. R., et.al "Surveying deep-sea hydrothermal vent plumes with the Autonomous Benthic Explorer (ABE)", Proc 12th UUST, Durham, New Hampshire, August, 2001
[6] Thomas, H., et.al, "Mapping AUV Survey of Axial Seamount", Eos Trans. AGU, 87(52), Fall Meet. Suppl., Abstract V23B-0615, 2006
[7] Bellingham, J. G., et.al "Field Results for an Arctic AUV Designed for Characterizing Circulation and Ice Thickness" AGU, Fall 2002.
[8] Carreras M., et.al, "Behaviour Control of UUV's" in Advances in Unmanned Marine Vehicles, by G. N. Roberts, Robert Sutton Eds, IEE Control Series, January, 2006.
[9] Brooks, R. A. "A robust layered control system for a mobile robot" IEEE Journal of Robotics and Automation, RA-2:14–23, 1986.
[10] Jonsson, A., P. Morris, N. Muscettola, K Rajan, B Smith "Planning in Interplanetary Space: Theory and Practice" AIPS 2000, Brekenridge.
[11] Frank, J., A. Jonsson, "Constraint-based attributes and interval planning," Journal of Constraints, vol. 8, Oct. 2003.
[12] Nilsson, N., "Teleo-Reactive Programs for Agent Control," Journal of Artificial Intelligence Research, 1:139-158, January 1994.
[13] Gertler, M., Hagen, G.R. "Standard Equations of Motion for Submarine Simulation" Naval Ship Research and Development Center Report 2510, June 1967.
[14] Muscettola N., et.al. "Remote Agent: To Boldly Go Where No AI System Has Gone Before" AI Journal 103(1-2):5-48, August 1998.
[15] Rajan K., et.al, "Remote Agent: An Autonomous Control System for the New Millennium," PAIS, 14th ECAI, Berlin, 2000.
[16] Muscettola N., et.al, "IDEA: Planning at the core of autonomous reactive agents", in Proc IWPSS, Houston, October 2002.
[17] Finzi, A., F. Ingrand, N. Muscettola, "Model-based Executive Control through Reactive Planning for Autonomous Rovers", IROS 2004
[18] Chien, S. et.al, "Using Autonomy Flight Software to Improve Science Return on Earth Observing One", J. of Aerospace Computing, Information Communication. April 2005
[19] Ingrand, F., et.al, "Decisional Autonomy of Planetary Rovers," Journal of Field Robotics, Vol 24, Issue 7, Pages 559 - 580, July 2007.
[20] D. Bernard, et.al, "Remote Agent Experiment: Final Report," NASA Technical Report, Feb. 2000
[21] Dechter, R., I. Meiri, J. Perl "Temporal constraint networks", Artificial Intelligence, Volume 49, Issue 1-3, pp 61 – 95, 1991
[22] Rosenblatt, J., et.al "A Behavior-based Architecture for Autonomous Underwater Exploration" Intnl. J. of Info Sciences, vol. 145, 2002.
[23] Turner, R. M., Stevenson, R. A. G. "ORCA: An adaptive, context-sensitive reasoner for controlling AUVs". in Proc 7th UUST, 1991.
[24] McEwen, R and Streitlien, K "Modeling and Control of a Variable-Length AUV" Proc 12th UUST, 2001.
[25] http://www.mbari.org/autonomy/TREX/index.htm