A Systematic Agent Framework for Situated Autonomous Systems

Frédéric Py
Monterey Bay Aquarium Research Institute
Moss Landing, California
fpy@mbari.org

Kanna Rajan
Monterey Bay Aquarium Research Institute
Moss Landing, California
kanna.rajan@mbari.org

Conor McGann
Willow Garage
Menlo Park, California
mcgann@willowgarage.com

ABSTRACT

We present a formal framework of an autonomous agent as a collection of coordinated control loops, with a recurring sense, plan, act cycle. Our framework manages the information flow within the partitioned structure to ensure consistency in order to direct the flow of goals and observations in a timely manner. The resulting control structure improves scalability since many details of each controller can be encapsulated within a single control loop. This partitioned agent design promises a domain-independent, scalable and robust approach for control of real-world autonomous robots operating in dynamic environments. We validate our framework with experimental results from deployments in two different real-world domains.

Categories and Subject Descriptors

I.2.9 [**Autonomous vehicles**]:

General Terms

Design

Keywords

Robot Planning, Cognitive robotics, Reactive vs deliberative approaches

1. INTRODUCTION

Autonomous robotic explorers must be proactive in the pursuit of goals and reactive to evolving environmental conditions. These concerns must be balanced over short and long term horizons to consider timeliness, safety and efficiency, presenting a substantial challenge for control system design.

As an illustrative example, consider an analogy in human control of a Remotely Operated Vehicle (ROV) for underwater exploration [4] to identify, sample and track a feature of interest such as an algal bloom. Typically the ROV is deployed from a ship with the following personnel acting specific roles as shown in Fig. 1:

Cite as: A Systematic Agent Framework for Situated Autonomous Systems, F. Py, K. Rajan, C. McGann, *Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, van der Hoek, Kaminka, Lespérance, Luck and Sen (eds.), May, 10–14, 2010, Toronto, Canada, pp. 583-590

- The *Scientist* has the overall knowledge of the science needs and drives the mission plan via high level directives such as where to go and what tools and sensors to use.
- The *Pilot* follows instructions while respecting operational constraints and tries to command the vehicle for a successful mission completion.
- The *Arm operator* is looking for items of scientific interest to collect and deposit in a sample tray taking commands from the *Scientist* or *Pilot*.
- Finally the ROV itself receives and executes the commands received (overall control from the Pilot and arm actuation from the Arm Operator) providing force feed-back or moving through the water column.

Often the scientist specifies the survey objectives in an abstract manner such as:

> Explore the box defined by the points *NE, SE, NW, SW* looking for a bloom feature with a minimum sampling resolution of *1km* and maximum resolution of *250m* between survey transects. Take up to *8* water samples within the feature preferably with a separation of *1km* from each other.

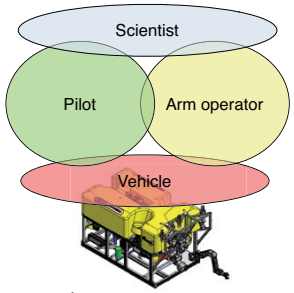

Figure 1: A conceptual view of actors in ROV operations.

Even though it may appear at first as providing substantial information, this objective is abstract; the vehicle needs to obtain sensor data to identify the bloom and to find the sequence of commands to send for efficient exploration of the defined area. Finding an episodic phenomenon such as a bloom in of itself has substantial uncertainty; doing so within the context of a mission that can include pursuing opportunistic goals while reacting to unexpected events that may threaten vehicle safety, is challenging.

Each actor has a different functional role over differing temporal scopes during the mission even as their shared knowledge is well defined. They are also, for the most part, entities with some deliberation and reaction capabilities. For instance, while the *Pilot* may need to react immediately to a potential navigation problem, the *Scientist* may need time to deliberate in order to alter mission objectives in the light of new information. In effect these actors represent individual sense-plan-act loops with their own foci and view of the

world, while interacting with each other in order to complete a mission.

We use this paradigm to motivate our agent design in an exploration context which can also be found in military, aviation and spaceflight operation domains. Such a command and control process suggests that responsibilities can be *partitioned* to exploit differing task definitions, abilities and functional and temporal scope for autonomous agent control.

We therefore, formally define an agent control structure as a composition of coordinated control loops with a recurring sense-plan-act (SPA) cycle. We manage the information flow within the partitioned structure to ensure consistency in order to direct the flow of goals and observations in a timely manner. The resulting control structure improves scalability since many details of each controller can be encapsulated within a single control loop. Furthermore, partitioning increases robustness since controller failure can be localized to enable graceful system degradation, making this an effective divide-and-conquer approach to the overall control problem. The role of our agent is to ensure that all the reactors will be able to interact concurrently so that they:

- are informed of state evolution that may impact them
- have a sufficient amount of time to synthesize plans
- coordinate plan dispatch across reactor boundaries

The dominant approaches for building situated robotic agent control systems, utilize a three-layered architecture [9], notable examples of which include IPEM [3], ROGUE [10], the LAAS Architecture [1], the Remote Agent Experiment [19] and ASE [6] (see [12] for a survey). Scalability is of concern since the planning cycle in these approaches is monolithic often making fast reaction times impractical when necessary; many of these systems also utilize very different techniques for specifying each layer in the architecture resulting in duplication of effort and a diffusion of knowledge ([19] and [6] are exemplars). With lessons learned from the Remote Agent experience, IDEA [7] was designed with interleaved planning and execution with a collection of controllers within a common framework. However IDEA, did not enforce a systematic framework for formally governing these interactions. In our view both the synchronization of state and dispatching of plan primitives are critical to ensure a correct behavior of the agent and in our view, fundamental to making the approach effective in practice. This is the novelty of our work.

The remainder of this paper is organized as follows. Section 2 gives a conceptual view of our framework to set the context of the analysis that follows in section 3. The latter is the core of the paper and describes the semantics of information exchange and deliberation. Section 4 highlights the real-world examples in the terrestrial and underwater domains with situated robots. We conclude with future plans in section 5.

2. A CONCEPTUAL VIEW OF A REACTOR

We conceptualize the reactor as a system that exhibits it's own *internal* state that is dependent on *external* observations from other reactors. Such information exchange occurs by sharing *state variables* over a common temporal horizon. A state variable describes the evolution of an attribute of the agent over time. For example, the Position of

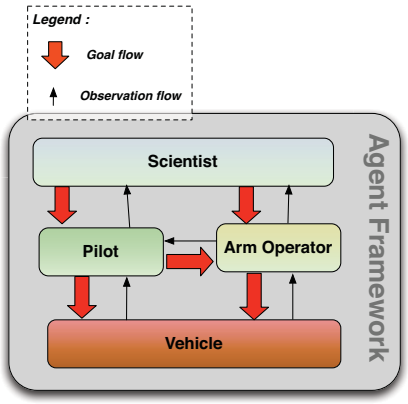

Figure 2: An agent is composed of multiple reactors or control loops.

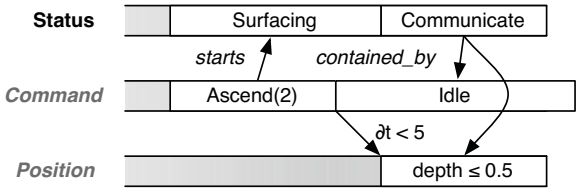

Figure 3: The *Pilot*'s plan for communication: Status is *internal* while Command and Position are *external* to this reactor.

The internal state of a reactor is driven in large part by the external environment. So to identify the real value of its *internal* state variable Status, the *Pilot* needs to know the current value of the two *external* state variables provided by another reactor, *Vehicle*. In our framework we allow only the reactor having an *internal* state variable as having the capability to model its evolution. It then becomes important that this state is correctly computed and in turn constrains other reactors that depend on its value. This operation, called *synchronization*, needs to be done as time advances in order to ensure that all reactors have the same view of state variables of the agent at least up to the execution frontier.

Conversely, the external world is influenced by partial plans generated by a reactor. A reactor's future objectives impact the evolution of its *external* state variables. If the *Pilot* has the goal to `Communicate` in the near future and the current position observed indicates a depth of $10m$ the *Pilot*

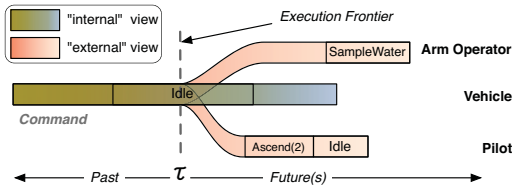

Figure 4: Reconciling different views of the Command state variable by the owner Vehicle.

needs to generate goals to appropriately constrain the Position and Command state variables. This operation, called *deliberation*, is intrinsic to the reactor. Accordingly, the *look-ahead* planning window and *latency* are parameters to help the agent identify when to prioritize a reactor's deliberation.

After deliberating the *Pilot* could produce a partial plan as shown in Fig. 3. It enters the state `Surfacing` which in turn requires the *external* state variable Command to be in state `Ascend(depth)` with $depth \leq 0.5$. `Ascend` is encapsulated within *Vehicle* which needs to be informed of these new objectives tasked by the *Pilot*. This operation, called *dispatching*, is undertaken to integrate the *external* state variables in a reactor's partial plan as goals for reactors owning these state variables.

Fig. 4 illustrates what appears to be divergent views of a single state variable, Command, getting reconciled by the owner reactor. For example, while the *Pilot* might plan as shown in Fig. 3, the *Arm Operator* might have planned to achieve the state to `SampleWater` while the *Vehicle* reactor might project the future state to be `Idle`. These views are not necessarily incompatible; we may be able to sample after `Idle`-ing, for instance. In the event of a conflict, the final arbitration is left to the reactor owning the state variable. Reconciliation of such divergent views necessitates the need for efficient coordination which requires our framework to enforce:

Synchronized state views: Each reactor needs to share the same view of the past and the current state of a state variable. This implies that when the *internal* state variables are updated, reactors that observe them as *external* state variables, need to be informed of state change so all reactors have a consistent view of the past and the present.

Exchange of intentions: A constraint *externally* applied on the future evolution of a state variable has to be integrated as an objective (or goal) to its *internal* representation, necessary for task delegation.

When not reconcilable, conflicting goals are rejected. These properties need to be maintained during the entire execution cycle of the agent while allowing reactors to deliberate within a reasonable time frame. To enforce these properties, we make two assumptions:

- Uniform tick rate: the world evolution is at a given frequency. All the state variables maintained inside an agent evolve at a rate that cannot be higher than this frequency. We do so to reduce complexity in bookkeeping and to ensure synchronous system state evolution.
- The past is monotonic: Observations produced at a given time by a reactor are statements of truth and cannot be changed in the future. This too is enforced to reduce system complexity.

3. AGENT ARCHITECTURE

3.1 Definitions

Our agent is a collection of reactors evolving as time advances and defined as the tuple $\{\mathcal{H}, \mathcal{R}, \mathcal{S}, \mathcal{L}, \mathcal{I}_{\mathcal{R}}, \mathcal{E}_{\mathcal{R}}, \mathsf{view}\}$ where :

- $\mathcal{H} = [0, D] \subseteq \mathbb{N}$ defines the interval of time during which the agent will be active; D represents the *lifetime* of the agent. In this context we define the following:
  - $\tau \in \mathcal{H}$ is the execution frontier of the agent depicting elapsed execution time.
  - A *tick* is a unit of time and has a fixed duration of δ_{tick}.
- $\mathcal{R}$: A finite set of reactors of the agent. Each reactor r provides information about:
  - its deliberation *latency* ($\lambda_r \in \mathbb{N}$) which indicates the maximum number of ticks it will need to produce a new plan.
  - and its deliberation *look-ahead* ($\pi_r \in \mathbb{N}$) indicating how far ahead it is deliberating.
- $\mathcal{S}$: The set of state variables in an agent.
- $\mathcal{L}$: The set of all the possible evolutions of the state variables of $\mathcal{S}$ over $\mathcal{H}$. The subset of this domain for a specific variable s is given as $\mathcal{L}_s$.
- $\mathcal{I}_{\mathcal{R}} : \mathcal{R} \rightarrow 2^{\mathcal{S}}$ A partition of $\mathcal{S}$ over $\mathcal{R}$ giving the mapping between reactors and their *internal* state variables. This means that for each state variable s there's one and *only one* reactor that declares it as *internal*:

$$\mathcal{S} = \bigcup_{r \in \mathcal{R}} \mathcal{I}_r \tag{1}$$

$$\forall (r_1, r_2) \in \mathcal{R} \times \mathcal{R} : \mathcal{I}_{r_1} \cap \mathcal{I}_{r_2} \neq \emptyset \Rightarrow r_1 = r_2 \tag{2}$$

- $\mathcal{E}_{\mathcal{R}} : \mathcal{R} \rightarrow 2^{\mathcal{S}}$ which indicates what *external* state variables a reactor r depends on. An *external* state variable of r cannot be *internal* to the reactor:

$$\forall r \in \mathcal{R} : \mathcal{I}_r \cap \mathcal{E}_r = \emptyset \tag{3}$$

- $\mathsf{view} : \mathcal{H} \times \mathcal{S} \times \mathcal{R} \rightarrow \mathcal{L} \cup \{\perp\}$ is a function that gives the current evolution of a state variable s as viewed by r at τ, otherwise it is symbolized by the special value $\perp$ indicating that r does not interact with s.

$$\forall (r, s) \in \mathcal{R} \times \mathcal{S} : s \notin \mathcal{I}_r \cup \mathcal{E}_r \Leftrightarrow \left(\forall \tau \in \mathcal{H} : \mathsf{view}_\tau(s, r) = \perp\right) \tag{4}$$

$$s \in \mathcal{I}_r \cup \mathcal{E}_r \Rightarrow \left(\forall \tau \in \mathcal{H} : \mathsf{view}_\tau(s, r) \in \mathcal{L}_s\right) \tag{5}$$

3.2 State variable representation

In our framework the view of a state variable evolution in a *reactor* is represented by *timelines*, a flexible representation to describe one sequence of states for a given state variable [18, 11, 14].

DEFINITION 1. *For each state variable $s \in \mathcal{S}$, timeline values $l \in \mathcal{L}_s$ of s are defined by the tuple $\{\mathcal{T}_s, \mathcal{Q}_l, \mathcal{G}_l\}$ where:*

- *$\mathcal{T}_s$ is the set of all the possible* tokens *for s. Each token $T \in \mathcal{T}_s$ expresses a constraint on the value of s over a temporal scope. It is described as the predicate*

$$T = p(\mathsf{start}, \mathsf{duration}, \mathsf{end}, \overrightarrow{x})$$

where p is the predicate name; attributes start, duration *and* end *are intervals over $\mathbb{N}$ indicating temporal scope during which p will hold. $\vec{x}$ describes the attributes of p with their acceptable domains. For example the token:*

$$\mathsf{Speed}(\ \mathsf{start} = [0,5], \mathsf{duration} = [1,10], \mathsf{end} = [1,15], \\ \mathit{speed} = [1, 1.5]\quad)$$

represents the speed of a vehicle between 1 and $1.5m/s$ starting between the tick 0 and 5 for up to 10 ticks.

- $\mathcal{Q}_l = \{T_1, \ldots, T_n\} \in 2^{\mathcal{T}_s}$ *is a sequence of valid* tokens *that describe a possible evolution of s. No two tokens $T_i \in \mathcal{Q}_l$ can be concurrent and they are sorted according to their relative* temporal *order. Formalizing using Allen algebra [2]:*

$$\forall i \in [1, n) : T_i \text{ meets } T_{i+1} \vee T_i \text{ before } T_{i+1} \quad (6)$$

- $\mathcal{G}_l \subseteq \mathcal{T}_s$ *describes the set of goals for this timeline. A goal is a token that needs to be evaluated for potential insertion in $\mathcal{Q}_l$ or rejected if insertion is not possible [8]. The result of deliberation is to find a way to make this goal set empty.*

Fig. 3, for example, shows `Communicate` and `Surfacing` tokens on the Status timeline.

3.3 Synchronization: Maintaining Agent state

To ensure that in our lock-step approach to maintaining the same view of agent state via state variables, we *synchronize* reactors.

Each reactor has its own view of a state variable. These views can be divergent and need to be aligned with a common ground truth at the execution frontier τ. The reference of this ground truth is the view provided by the reactor declaring this state variable as *internal* ($s \in I_r$). The agent's role is then to ensure that this value can be correctly computed and distributed to reactors observing this state variable. The sequencing of this operation is crucial as an *internal* state variable depends on the state of the *external* state variable of this reactor. We define this operation as follows:

Definition 2. *For each state variable $s \in \mathcal{S}$ a* view *of $l \in \mathcal{L}_s$ is synchronized at τ, $\text{Ⓢ}(l, \tau)$, when its value up to τ reflects the evolution of s.*

By extension a reactor r is synchronized at τ ($\text{Ⓡ}(r, \tau)$) iff all its internal *state variables views are* synchronized*:*

$$\forall r \in \mathcal{R} : \\ \text{Ⓡ}(r, \tau) \leftrightarrow \big(\forall s \in \mathcal{I}_r : \text{Ⓢ}(\mathsf{view}_\tau(s, r), \tau)\big) \quad (7)$$

To maintain agent state, we ensure that all the reactors are synchronized each time τ advances. Doing so guarantees that all reactors have a common view of the world up to the execution frontier τ. In our framework the *internal* state variable values depends on the *external* state variables. Fig. 5 illustrates this relationship for the *Pilot*.

For example, in order to identify that it is in the `Communicate` state on its *internal* Command state variable, the *Pilot* needs to be aware that that the Position is close to the surface and the Command currently executed is *Idle*. Both these state variables are owned by the *Vehicle*. As a result the *Pilot* cannot deduce its *internal* state before having a correct view of the *external* state variables it relies on.

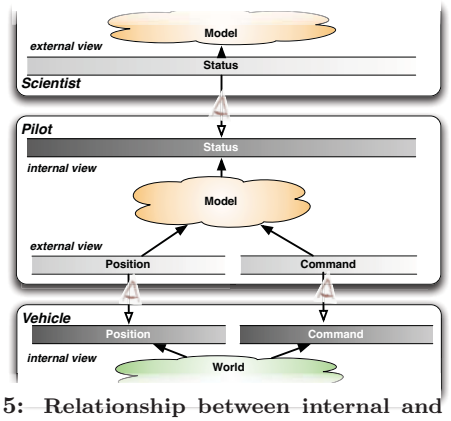

Figure 5: Relationship between internal and external views **of a state variables. The internal** view **is deduced based on the reactor's model; the external** view **"observes" the state variables owned by a different reactor.**

Lemma 3.1. *At τ for the reactor r, the* internal view *of a state variable $s \in \mathcal{I}_r$ cannot be synchronized before its state variables in $\mathcal{E}_r$ are synchronized:*

$$\forall r \in \mathcal{R}, \forall s \in \mathcal{I}_r : \\ \text{Ⓢ}(\mathsf{view}_\tau(s, r), \tau) \Rightarrow \big(\forall e \in \mathcal{E}_r : \text{Ⓢ}(\mathsf{view}_\tau(e, r), \tau)\big) \quad (8)$$

Corollary 3.2. *A reactor cannot be synchronized before all its* external views *of state variables are.*

Proof. This follows from the definition of Ⓡ on Equation 7 and the Lemma 3.1. □

Definition 3. *The maximum synchronization duration per reactor r is $\delta^r_{sync} \in \mathbb{R}$.*

The synchronization between *external* and *internal* state variables is the responsibility of the reactor. As stated in Corollary 3.2 this operation can only be done after *external* state variables of r are synchronized. Further, since the correct state of an *external* view is produced by its corresponding *internal* view in a reactor which has ownership, we show:

Lemma 3.3. *At any time τ for reactor r_1, an* external view *of a state variable $s \in \mathcal{E}_{r_1}$ is synchronized if and only if the corresponding* internal view *on reactor r_2, $s \in \mathcal{I}_{r_2}$ ($r_2 \neq r_1$) is synchronized and both* views *are identical up to τ.*

Such relationships imply a natural dependency chain between reactors that can be defined as follows:

Definition 4. *A reactor r_1* depends *on another reactor r_2, shown as $r_1 \rhd r_2$, iff some subset of its* external *state variables are* internal *to r_2. Formally:*

$$\forall (r_1, r_2) \in \mathcal{R} \times \mathcal{R} : \\ r_1 \rhd r_2 \Leftrightarrow \mathcal{E}_{r_1} \cap \mathcal{I}_{r_2} \neq \emptyset \quad (9)$$

We can reduce the synchronization problem as follows:

Corollary 3.4. *A reactor cannot be synchronized before all the reactors it depends on are synchronized.*

$$\forall (r_1, r_2) \in \mathcal{R} \times \mathcal{R} : \\ \text{Ⓡ}(r_1, \tau) \wedge r_1 \rhd r_2 \Rightarrow \text{Ⓡ}(r_2, \tau) \quad (10)$$

We can then affirm the following:

THEOREM 3.5. *To ensure synchronization convergence of all reactors of the agent, cyclic dependency between reactors should be prohibited.*

PROOF. Assume two reactors r_1, r_2 have a cyclic dependency such that $r_1 \rhd r_2$ and $r_2 \rhd r_1$. Corollary 3.4 implies that none of them can be synchronized before the other is. Consequently the only way to have these two reactors synchronized is to search for a fixed point until both appear to maintain their state value. A fixed-point iteration in general is not guaranteed to converge, therefore we cannot guarantee that these reactors will eventually be synchronized to stabilize overall agent state. □

By enforcing no cyclic dependency we can find a linear order between the reactors which ensures that a if $r_1 \rhd r_2$ then r_1 won't be synchronized by the agent until r_2 is. Synchronization between reactors then will result in a single pass iteration through all the reactors which is $O(\|\mathcal{R}\|)$ provided that δ^r_{sync} (the maximum synchronization time) $\forall r \in \mathcal{R}$ are bounded. As a result the synchronization time complexity is linear in the number of reactors.

To guarantee that all the reactors are synchronized at each tick for our agent:

$$\sum_{r \in \mathcal{R}} \delta^r_{sync} \leq \delta_{tick} \tag{11}$$

Eqn 11 would be sufficient if the reactors did not have to deliberate and just keep track world evolution. As we expect them to take time to deliberate this relation is superseded as:

$$\sum_{r \in \mathcal{R}} \delta^r_{sync} \ll \delta_{tick} \tag{12}$$

When all reactors respect Eqn 12 the agent has a correct view of the evolution of the world. This is a necessary basis for sound deliberation.

3.4 Dispatching: Ensuring common objectives

Reactors *dispatch* partial plans to task other reactors via *external* state variables. This is the primary mode of agent wide execution between reactors (as also to actuate robotic hardware). Understanding deliberation is key however, to knowing when precisely dispatching can occur.

A reactor r deliberates to find a plan that fulfills goals attached to its *internal* or *external* views $\mathcal{G}_{\mathsf{view}_\tau(r)}$. Deliberation results in the reduction of this goal set to $\emptyset$ by attempting to insert the goal in the plan (or by rejecting it). Even though a reactor may need more than one tick to produce its plan, the agent can interrupt this process at any time without prohibiting the reactor to synchronize its current state.

DEFINITION 5. *We define the expression* $\mathsf{deliberate} : \mathcal{H} \times \mathcal{R} \rightarrow \{\top, \bot\}$ *indicative of a reactor's need to deliberate. This expression is necessarily true if any of the views of the reactor have a non empty goal set :*

$$\forall r \in \mathcal{R}, \forall \tau \in \mathcal{H} :$$
$$\mathsf{deliberate}_\tau(r) \Leftarrow \left(\exists s \in \mathcal{I}_r \cup \mathcal{E}_r : \mathcal{G}_{\mathsf{view}_\tau(s,r)} \neq \emptyset\right) \tag{13}$$

As soon as a reactor has an objective attached to at least one of its *internal* state variables it needs to deliberate. Using the definition of the deliberation latency (λ_r) for reactor r from section 3.1 the following property must hold for deliberation to occur within r:

PROPOSITION 3.6. *The maximum duration during which a reactor r is in deliberation, is its deliberation latency λ_r.*

At the end of deliberation a reactor is expected to have a partial plan. This plan potentially includes tokens attached to the evolution of its *external* state variables. Fig. 3 illustrates how the `Communicate` token on the Status state variable, sub-goals to `Idle` and `depth` on the Command and Position *external* state variables respectively. This view of the future is pending and needs to be reconciled during synchronization.

DEFINITION 6. *For each* external *state variable s of a reactor r, we define the set* $\mathsf{pending}_\tau(s, r)$*, defined only when the reactor is not deliberating, to include all tokens that describe the desired evolution of these state variables for this reactor's plan.*

$$\forall (\tau, r) \in \mathcal{H} \times \mathcal{R}, \neg\mathsf{deliberate}_\tau(r), \forall s \in \mathcal{E}_r :$$
$$pending_\tau(s, r) = \{t \in \mathcal{Q}_{\mathsf{view}_\tau(s,r)} : t.\mathsf{start} > \tau\} \tag{14}$$

Indeed, reactor deliberation during and after, can alter this future. By extension we define the overall agent $\mathsf{pending}_\tau(s)$ *set as:*

$$\mathsf{pending}_\tau(s) = \bigcup_{r \in \mathcal{R}} \mathsf{pending}_\tau(s, r) \tag{15}$$

The role of the agent then, is to provide an efficient mechanism to transfer these pending tokens as goals for the *internal* views of corresponding state variables. If these goals are feasible the reactor will ensure that the corresponding tokens are now part of the *internal* view of the future evolution of the reactor.

PROPOSITION 3.7. *The goal set of internal state variables of a reactor r is a subset of* pending *tokens of the external* views *of these state variables in reactors depending on r.*

$$\forall r \in \mathcal{R}, \forall s \in \mathcal{I}_r :$$
$$\mathcal{G}_{\mathsf{view}_\tau(s,r)} \subseteq \mathsf{pending}_\tau(s) \tag{16}$$

As the reactor (and by extension agent) state is evolving, and synchronization and deliberation are to occur between *ticks*, a key question is, when can a pending token be included in a state variable's goal? Each reactor r, provides a priori, its maximum deliberation latency λ_r. In addition the reactor must also account for sufficient time to ensure that the *external* views of its partial plan can be reconciled into other reactors.

DEFINITION 7. *A reactor's execution latency (Λ_r) expresses the time necessary for a reactor to both deliberate and correctly dispatch the outcomes of its partial plan on its* external *state variables. It is recursively defined as:*

$$\Lambda_r = \lambda_r + \max_{r' \in R : r \rhd r'} (\Lambda_{r'}) \tag{17}$$

Using the look-ahead π_r, we can then identify the planning window as a temporal duration over which reactor r deliberates:

THEOREM 3.8. *When a reactor starts its deliberation at τ its planning window is specified as:*

$$\Pi_r(\tau) = [\tau + \Lambda_r, \tau + \Lambda_r + \pi_r] \cap \mathcal{H} \tag{18}$$

PROOF. Let $r_1 \rhd r_2$. Consider that reactor r_1 takes its full latency λ_{r1} to produce its partial plan. It would then be able to dispatch its pending tokens only after $\tau + \lambda_{r1}$.

Now consider that reactor r_2 also uses its full latency to produce a plan for one of these pending tokens t_p. The observation of t_p during synchronization of r_1 will not occur before $\tau + \lambda_{r1} + \Lambda_{r2}$. Consequently to ensure a viable execution of the plan resulting from deliberation, reactor r_1 should plan on a window starting from $\tau + \Lambda_{r1}$. Therefore the upper bound of r_1's planning window is given by: $\tau + \Lambda_{r1} + \pi_{r1}$. □

Using this information from each reactor, the agent can now efficiently identify which pending tokens need to be included in corresponding *internal* goal sets. Proposition 3.7 can be refined as:

PROPOSITION 3.9. *For each state variable s the pending tokens that can be considered for deliberation by the com-*

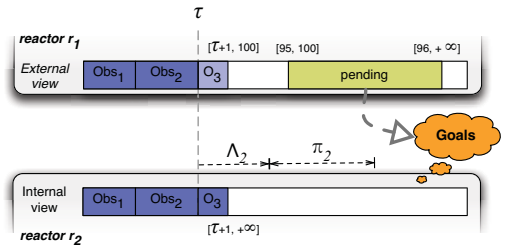

Figure 6: Transfer of a pending token to its corresponding goal set. Reactor r_1 has a pending token overlapping the planning window of r_2 to be inserted in its goal set.

In Fig. 6, as soon as a pending token of state variable s ($s \in \mathcal{E}_{r_1}$), can start within the planning window of r_2 which owns s, the agent inserts this token in the current goal set for r_2 to deliberate.

pending tokens that necessarily start before $\tau + \Lambda_r$ cannot be safely included in reactor r's goal set, since r may need its full latency to produce its partial plan and therefore won't be in a position to integrate this goal in time. Conversely pending tokens starting necessarily *after* $\tau + \Lambda_r + \pi_r$ will not be inserted in the goal set as they won't be in the planning window of the reactor.

3.5 An illustrative example

We have so far identified two ways in which the state information flows between reactors:

- Synchronization allows reactors to deduce their state so as to reconcile their views.
- Goal dispatching allows a reactor to propose new objectives to reactors it depends on.

When a goal is posted to a reactor r there are no guarantees that it can be inserted and planned by r. It can be rejected due to conflicts with its existing partial plan or its execution can be interrupted by an unanticipated situation. This can impact the plan of the reactor that initially posted the goal. To answer the question of how the reactor observes an invalidation of a plan and measures the impact of this failure, we return to the example from Section 2 with the *Pilot* having to `Communicate`.

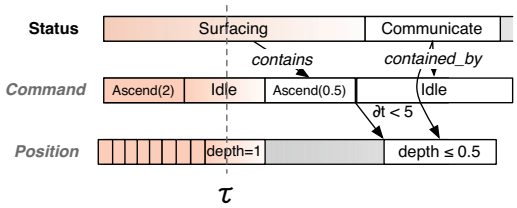

Figure 7: A partial plan produced by the *Pilot* after an initial plan failure based on Fig. 3. Red areas before τ indicate past observations grounded by synchronization.

As shown in Fig. 3, in this plan the *Pilot* has produced tokens which are attached to its *external* state variables owned by the *Vehicle* reactor. The *Pilot* proposes that *Vehicle* execute an `Ascend(2)` on the Command state variable, which would task the vehicle to ascend gradually, first up to the depth of 2m followed by an `Idle` and then to be able to reach a depth below 0.5m with a maximum delay of 5 ticks (such incremental ascends are done to disallow breaching).

This plan produces pending tokens which are dispatched to the goal set of *Vehicle*. `Ascend` is correctly inserted and its execution started by the *Vehicle* reactor. This state change is then synchronized back with the *Pilot* which in turn can identify that the state of the Status state variable has changed to `Surfacing` as expected. At time t the *Pilot* is notified about the termination of `Ascend` and the beginning of `Idle`. The *Pilot*'s partial plan requires the depth to be less than 0.5m within the next 5 ticks.

Let us assume that an external condition has altered the vehicle's buoyancy. As a consequence the vehicle does not rise as predicted by the model and the depth observed at tick $t + 5$ is around 1m below the surface. At synchronization time the *Pilot* identifies that this observation on the Position state variable is conflicting with its initial partial plan. However the *Pilot* needs to fulfill its objective to `Communicate`. Synchronization *breaks* the current plan to trigger a new deliberation phase to replan from an unexpected situation. The *Pilot* retracts the `Idle` and replans.

The *Pilot* can now safely replan knowing that the previously pending tokens are no longer part of the *Vehicle*'s goal set. A potential recovery action is to try to `Ascend` again but with a goal depth of 0.5m or less to ensure that the vehicle will be at the surface at the end as shown in Fig. 7. The agent should then be able to `Communicate` as planned.

PROPOSITION 3.10. *When synchronization invalidates the current plan of a reactor it potentially invalidates all its pending tokens on its* external *state variables. Consequently the goals on the corresponding* internal view *should be removed from this goal set.*

4. EXPERIMENTAL RESULTS

Our framework, called the **T**eleo-**R**eactive **EX**ecutive (`T-REX`) has been demonstrated in two challenging problem settings; a terrestrial hallway navigation domain requiring coordination between different reactors dealing with manipulation and control. And a marine robotic vehicle navigating, detecting and adapting to dynamic features in a complex coastal environment. Our implementations involved the use of the EUROPA$_2$ Constraint-based temporal planner [11, 5] for deliberation. Desktop experiments primarily to validate the notion of partitioning are presented in [14].

4.1 Service Robotics Example

PR2 is a mobile manipulation platform designed for operation in dynamic and unstructured indoor environments. In June, 2009, a series of demonstrations were conducted with a *PR2* requiring autonomous navigation, door-opening, and recharging using standard electrical outlets [13]. Notably, a critical demonstration required 10 recharge goals to be accomplished consecutively without any human intervention, in under an hour. In this and other demonstrations, T-REX gracefully handled a wide range of failure conditions.

Figure 8: *PR2* plugging in to a standard outlet.

Table 1 provides a summary based on a single run with 9 recharge goals (based on the number of offices available during experimentation). T-REX operated at 10 Hz with 7 reactors. This provides rapid transitions from one action to another but requires synchronization of the complete agent once every 100 ms. The overall scale of the system is indicated by the number of internal and external timelines. The total number of EUROPA$_2$ timelines excludes these. The mission ran for over an hour. 3 doors were locked when tried. The agent deferred them till later. One of them was opened when revisited. The remaining 2 were continually retried until the test was terminated. In total, 494 external robot actions were executed, of which 29 aborted or timed-out. 907 planning cycles occurred across all reactors that deliberate. A planning cycle is initiated when a reactor receives a goal, or when a flaw is entailed by the model. T-REX memory consumption was steady at 10 MB. The total line count for the model provides a coarse metric of program complexity. It includes all constraints and class declarations. The most number reflects the leverage from automated planning and a very high level programming model. T-REX ran on a dual-core 2.6GHz Linux machine. The mean and standard deviations for CPU utilization are at 9.8% and 5.0% respectively. This indicates that even at a control rate of 10 Hz T-REX was comfortably able to handle the load.

4.2 Marine Robotics Example

T-REX has also been integrated on-board an autonomous underwater vehicle (AUV – shown in Fig. 9) and deployed for scientific exploration in coastal waters off California [1]. The agent runs on a 367 MHz EPX-GX500 AMD Geode stack using Linux, with the lower-level functional layer running on a separate processor on real-time QNX. Over the course of 2008-9, our science objectives were to carry out

Figure 9: An MBARI AUV at sea with its support ship the R/V Zephyr.

survey missions within a prescribed area in excess of 50 Sq. km while estimating the presence of a dynamic coastal feature[1].

Details of one such mission are shown in Table 2, with T-REX running at 1Hz. The AUV exhibited both long term planning as well as reactive behavior in response to environmental changes impacting the full scope of the mission. During the uninterrupted 6 hour and 40 minutes run, T-REX was able to to bring back targeted water samples from within biological hot-spots [16, 15]. The deployed agent had a 5-reactor hierarchical configuration with two reactors being interfaced to the vehicle control system and a satel-

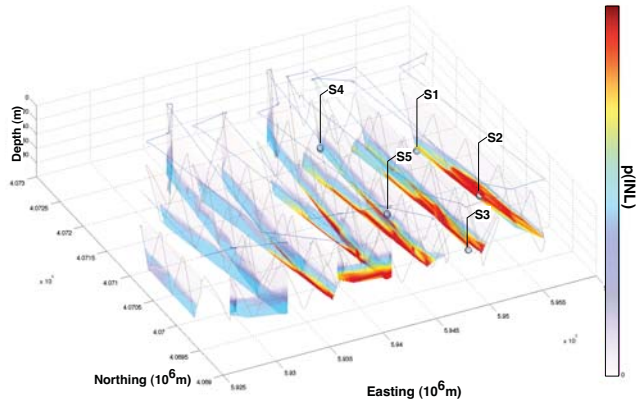

Figure 10: Visualization of a survey mission to detect and characterize a dynamic ocean feature. High probability of feature presence is indicated in red. S1-S5 indicate triggering of 10 water samplers.

Fig. 10 shows the AUVs transect and the context of the feature in the water-column detected by its sensors. It also shows the vehicle changing its sampling resolution, starting

[1]Fluid sheets of suspended particulate matter that originate from the sea floor.

Measurement	Value
Agent control rate ($1/\delta_{tick}$)	10 Hz
Total number of *internal* timelines	47
Total number of *external* timelines	66
Total number of EUROPA$_2$ timelines	87
Mission duration ($\mathcal{H}$)	3799 seconds
Estimated model line count	1207
T-REX CPU utilization (mean $\pm$ std)	$9.8\% \pm 5.0\%$

Table 1: Summary metrics for a service robot.

Measurement	Value
Agent control rate ($1/\delta_{tick}$)	1 Hz
Total number of *internal* timelines	21
Total number of *external* timelines	23
Total number of EUROPA$_2$ timelines	56
Mission duration ($\mathcal{H}$)	6:40 hours
Estimated model line count	2620
T-REX CPU utilization (mean $\pm$ std)	$14\% \pm 8.0\%$

Table 2: Summary metrics for a deployed marine robot.

with high and ending with low resolution transects where the feature is not visible. Water samples are shown to be taken within the feature as required. Scientific results from these missions in coastal larval ecology can be found in [20].

5. CONCLUSIONS AND FUTURE WORK

We define a novel formal framework which provides a basis for partitioning a complex control problem into multiple SPA control loops within an agent for seamless interaction. They do so by deliberating over their own state variables and allowing other control loops to observe and suggest goal sets for reconciliation. The strong semantics placed on state variable ownership and observation, allows the agent to systematically arbitrate to generate a consensus behavior via reactor deliberation.

One primary shortcoming with the framework, has to do with the inter-dependence between modeling of state variables and reactor design. State-Variables owned within a reactor encapsulate local interactions. When more than one user of a timeline attempts to post goals at the same time, it is problematic if there are interactions between these goals that are not captured in the owner reactor. Additionally, when timelines have co-temporal interactions, they should be reasoned about together in a single reactor. These considerations have not proven problematic in our experience. However, reactor design is still heuristically driven; techniques developed for partitioning constraint graphs such as [21] for example, may be relevant in this context.

Future areas of research are to understand the scalability of this approach by extending the agent's model deeper towards the hardware for diagnosis and recovery, to study how to extend our framework to multi-agent environments with sporadic communication capabilities in harsh environments and to study the capability to seamlessly insert and remove reactors within an agent, The latter as a way to deploy new capabilities as also to demonstrate fail-safe behavior in mission-critical settings.

6. ACKNOWLEDGMENTS

We thank the David and Lucile Packard Foundation, MBARI and WillowGarage for supporting our work. Additionally thanks to our colleagues Thom Maughan, Tom O'Reilly, Brent Roman, John Ryan, Chris Scholin, Hans Thomas, Bob Vrijenhoek and the crew of the R/V *Zephyr*. And to Andrea Orlandini and the annonymous reviewers for helpful comments on an earlier version of this paper.

7. REFERENCES

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *Intnl. Journal of Robotics Research*, Jan 1998.

[2] J. Allen. Towards a General Theory of Action and Time. *Artificial Intelligence*, 23(2):123154, 1984.

[3] J. Ambros-Ingerson and S. Steel. Integrating Planning, Execution and Monitoring. *Proc. 7th AAAI*, Jan 1988.

[4] R. Bachmayer, S. Humphris, and D. Fornari. Oceanographic Research Using Remotely Operated Underwater Robotic Vehicles: Exploration. *Marine Technology Society Journal*, Jan 1998.

[5] J. Bresina, A. Jonsson, P. Morris, and K. Rajan. Activity Planning for the Mars Exploration Rovers. In *ICAPS*, Monterey, California, 2005.

[6] S. Chien, R. Knight, A. Stechert, R. Sherwood, and G. Rabideau. Integrated Planning and Execution for Autonomous Spacecraft. *IEEE Aerospace*, 1:263–271 vol.1, 1999.

[7] A. Finzi, F. Ingrand, and N. Muscettola. Model-based Executive Control through Reactive Planning for Autonomous Rovers. In *Proc. International Conference on Intelligent Robots and Systems*, 2004.

[8] J. Frank and A. Jónsson. Constraint-based Attribute and Interval Planning. *Constraints*, 8(4):339–364, 2003.

[9] E. Gat. On Three-Layer Architectures. In D. Kortenkamp, R. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 195–210. MIT Press, 1998.

[10] K. Haigh and M. Veloso. Interleaving Planning and Robot Execution for Asynchronous User Requests. *Autonomous Robots*, Jan 1998.

[11] A. Jónsson, P. Morris, N. Muscettola, K. Rajan, and B. Smith. Planning in Interplanetary Space: Theory and Practice. In *AIPS*, 2000.

[12] R. Knight, F. Fisher, T. Estlin, and B. Engelhardt. Balancing Deliberation and Reaction, Planning and Execution for Space Robotic Applications. *Proc. IROS*, Jan 2001.

[13] C. McGann, E. Berger, J. Boren, S. Chitta, B. Gerkey, S. Glaser, E. Marder-Eppstein, B. Marthi, W. Meeussen, T. Pratkanis, and M. Wise. Model-based, Hierarchical Control of a Mobile Manipulation Platform. In *4th Workshop on Planning and Plan Execution for Real World Systems, ICAPS*, 2009.

[14] C. McGann, F. Py, K. Rajan, and A. Olaya. Integrated Planning and Execution for Robotic Exploration. In *Intnl. Workshop on Hybrid Control of Autonomous Systems, in IJCAI09*, Pasadena, California, 2009.

[15] C. McGann, F. Py, K. Rajan, J. P. Ryan, and R. Henthorn. Adaptive Control for Autonomous Underwater Vehicles. In *AAAI*, Chicago, IL, 2008.

[16] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen. A Deliberative Architecture for AUV Control. In *ICRA*, Pasadena, CA, May 2008.

[17] C. McGann, F. Py, K. Rajan, H. Thomas, R. Henthorn, and R. McEwen. Preliminary Results for Model-Based Adaptive Control of an Autonomous Underwater Vehicle. In *Int. Symp. on Experimental Robotics*, Athens, Greece, 2008.

[18] N. Muscettola. HSTS: Integrating Planning and Scheduling. In M. Fox and M. Zweben, editors, *Intelligent Scheduling*. Morgan Kaufmann, 1994.

[19] N. Muscettola, P. Nayak, B. Pell, and B. Williams. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *Artificial Intelligence*, 103, 1998.

[20] J. P. Ryan, S. Johnson, A. Sherman, K. Rajan, F. Py, J. Paduan, and R. Vrijenhoek. Intermediate Nepheloid Layers as Conduits of Larval Transport. *Limnology & Oceanograhy: Methods*, 2010. Accepted for publication.

[21] M. A. Salido and F. Barber. Distributed CSPs by Graph Partitioning. *Applied Mathematics and Computation*, 183:212–237, 2006.