RADIC - A Generic Component for the Integration of Existing Reactive and Deliberative Layers

Matthias Scheutz
Artificial Intelligence & Robotics Lab
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
mscheutz@cse.nd.edu

James Kramer
Artificial Intelligence & Robotics Lab
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556, USA
jkramer3@cse.nd.edu

ABSTRACT

Hybrid architectures have been developed to preserve the responsiveness of reactive layers while also providing the benefits of higher level deliberative capabilities. The challenge of hybrid architecture design is to integrate layers of very different functional roles. We propose a component, called RADIC, that uses a generic technique for the construction of hybrid architectures from pre-existing reactive and deliberative layers that requires only minimal modifications to each layer. In particular, we give a high-level description of the operation and benefits of the component and its algorithms.

Categories and Subject Descriptors

I.2.9 [**Artificial Intelligence**]: Robotics

General Terms

Design

Keywords

hybrid architectures, architecture integration, robotics

1. INTRODUCTION

Hybrid architectures for autonomous robots combine reactive and deliberative layers to preserve both responsiveness and the efficiency of symbolic representations. Yet, as pointed out in [11], "hybridism usually translates to *ad hoc* or unprincipled designs with all its attendant problems" and "hybrid architectures tend to be very application-specific."

To address these issues, we present a Reactive And Deliberative Layer Integration Component (RADIC) that provides a generalized "bridge" between layers. By maintaining layer separation, RADIC allows their principled design in isolation; by effecting a mapping of data between layers of disparate functionality, RADIC provides a general connection method to reduce application related specificity.

AAMAS'06 May 8–12 2006, Hakodate, Hokkaido, Japan.

2. HYBRID ARCHITECTURES

For autonomous robotic agents, which require fast, highly responsive reactive layers (e.g., [7, 10]), the design of hybrid architectures is difficult, largely because the nature of the layers is so different. The challenge of hybrid architectures is to integrate the layers' disparate aspects of operation, at least achieving the following functional mappings:

F1: between "global" and "egocentric" relations
F2: from discrete actions to continuous motion
F3: between logical update time and real-time operation
F4: from stateful to stateless operation

We give a brief overview of a varied selection of hybrid architectures, describing how each satisfies items **F1-F4**:

***SSS* [4]:** Short for "*S*ervo, *S*ubsumption, *S*ymbolic" layers, the symbolic layer uses a coarse-grained world model that configures the reactive layer via parameterization of selected behaviors in the subsumption layer. SSS explicitly addresses the functional mappings by discretizing space between the servo and subsumption layers (**F1, F2**) and time between the subsumption and symbolic layers (**F3, F4**).

***AuRA* [2]:** AuRA's deliberative layer is composed of a planner, spatial reasoner, and sequencer, while the reactive layer consists of libraries of behavior schemas, activated by the sequencer and adaptable via a homeostatic component, producing actuation commands via vector summation of schema output. Functional mappings **F1**, **F3** and **F4** are made through the interaction of the schema controller with the plan sequencer, spatial reasoner, and planner, while **F2** is accomplished by the selection of behavior schemas.

***3T* [3]:** The 3T (short for "three tier") architecture, which extends work on both the RAPs [5] and ATLANTIS [6] architectures, uses a deliberative planner to synthesize goals in a partially ordered task list, executed by reactive skills (RAPs), bridged by a skill manager/sequencer (RAP interpreter). Tiers tend to operate at different time scales to fulfill mapping **F3**. Mappings **F1** and **F4** use all tiers; the planner operates at a high level of abstraction, the sequencer selects a set

of skills from the RAP library, while skills operate in a context dependent manner. Mapping **F2** is performed by the integration of skills and the sequencer; RAPs are symbolic, discrete steps for accomplishing a task composed of skills that implement continuous motion.

***Saphira* [8]:** Saphira uses a strong internal world model called the *local perceptual space* (LPS) that coordinates sensory data with representations. Planning is performed by the *PRS-Lite* component, which (de)activates sets of behaviors that are fused using fuzzy logic. Functional mappings **F1** and **F3** are accomplished via the LPS and PRS-Lite, while **F2** is accounted for by behavior selection and fuzzy logic fusion. Abstract representations are integrated in the LPS to fulfill mapping **F4**.

***4-D/RCS* [1]:** 4-D/RCS features a hierarchy of six levels (actuator, servo, primitive, subsystem, vehicle, and section) that operate on different scales. Each contains a supervisor and a set of subordinate agents that supervise units at the next lower level. Each node contains behavior generation, world modeling, sensory processing, value judgment processes and a knowledge database. Mappings **F1-F4** are explicitly made between each level.

While all of the above hybrid architectures successfully combines reactive and deliberative layers, each subscribes to a specific design philosophy, if not in integrating layers, then by specifying the design of the layers themselves. This restriction not only limits potential reuses of the developed layers and software, but more importantly the extent to which the designs themselves generalize. It would be desirable to achieve a more general connection or integration method that could be employed for many different reactive and deliberative layers without the need for much adaptation of either layer. Ideally, we would like a single, generic, "mediating component" to connect existing reactive and deliberative layers that successfully achieves the requisite functional mappings **F1-F4**. Several design desiderata suggest themselves for such a component; in particular, the method should:

I1: require minimal changes to existing layers

I2: provide integration of individual layers, while executing independently, autonomously, and in parallel

I3: use minimal processing time

I4: add functionality that improves the performance of systems that contain the layers in isolation

All the hybrid architectures above fail to satisfy some of **I1-I4**, in addition to the various criticisms (including those presented in [6, 3, 9]). Neither Saphira nor 4-D/RCS satisfies **I1**, albeit for different reasons. While the large amount of integration performed by the LPS in Saphira has the benefit of *coherency*, it also means that functional changes necessitate large modifications. On the other hand, the strict specification of interfaces between levels and substantial data flow within a single level in 4-D/RCS has the effect that slight modifications to functionality may entail large changes across components or levels. None of SSS, AuRA, and 3T meet **I2** because all switch completely back and forth between reactive and deliberative layers; the deliberative layer produces a fully formulated plan that is passed to the reactive layer, which then remains in control until a step either completes or fails. Both Saphira and 4-D/RCS suffer from difficulties with **I3**; the use of the LPS in Saphira requires a substantial amount of processing, while the full complement of components in 4-D/RCS introduces possibly redundant or unnecessary processing. Finally, **I4** presents difficulties for 3T, AuRA, and 4-D/RCS. In both 3T and AuRA, operation of the reactive layer is tightly knit and depends completely on the sequencer, making it difficult to add external functionality. A similar situation occurs with 4-D/RCS, where the functionality of a given level is specified completely by its internal operation. The proposed RADIC component, which we describe next, has been designed to satisfy all the functional mappings **F1-F4** and design desiderata **I1-I4**.

3. THE RADIC APPROACH

Functional mappings **F1-F4** are made in all hybrid architectures. The approach taken with RADIC can be described as effecting a *generic technique* for integrating (1) a *sequence of goals* originating from a deliberative layer, (2) a stream of *sensor data*, and (3) a *sequence of actions* (e.g., motor commands) from the reactive layer (see Figure 1), as performed by the `updateRADIC` algorithm given in Figure 2. The universality of a particular RADIC component is dependent on the representations used in mapping data between layers; at some application-dependent level, representations are either too abstract to be useful or too narrow to be applicable. For instance, a deliberative layer that produces a goal list composed of an ordered sequence of spatial locations to "visit" (such as in navigation or arm movement) requires translation of world-coordinates into egocentric-coordinates. While a RADIC component that integrates such layers may be applied to other tasks requiring spatial reasoning, it may not be adequate for, say, a higher level goal list that includes what to do at each location. RADIC provides a generic technique for integrating layers, but specificity is dependent on mapping implementation.

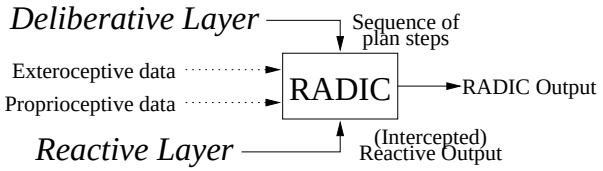

Figure 1: Functional diagram of the RADIC component showing (possible) inputs and output.

Functional mapping **F1** is satisfied by the `convertGoal` function, detailed below. The `updateRADIC` function is called at least as frequently as the reactive layer produces outputs (e.g., sends motor commands), and thus satisfies item **F3**. The functional mappings **F2** and **F4** are an inherent part of the RADIC component's operation: a goal is a discrete representation, while RADIC's output effects continuous motion (**F2**, either via discrete actions or via discrete motor commands), and the state of the deliberative layer is maintained in internal memory, while not requiring any state from the reactive layer other than its outputs (**F4**).

In more detail, the `updateRADIC` function takes as arguments the outputs from the reactive layer *R-out* (typically, intercepted motor commands), the changes as determined by sensors *S-change* (proprioceptive, exteroceptive, or missing), a sequence of goals from the delibera-

tive layer *D-goals* (which can be empty), and a strategy for goal attainment *G-strat*. The sequence of goals *D-goals* $= \langle G_1, G_2, ..., G_n \rangle$ is transformed, in accordance with item **F1**, from a "global" representation into an ego-centric representation $\langle \overline{G}_1, \overline{G}_2, ..., \overline{G}_n \rangle$ by the `convertGoal` function and stored in RADIC's internal memory M. `convertGoal` also associates a set of "blocking goals" B_G and a set of goal conditions C_G with each goal G that will depend on *G-strat* (in general, *G-strat* is a function that defines or influences the relation among G, B_G, and C_G). Blocking goals are used to impose a priority ordering on goals (goals that are blocked are not considered for action selection). Similarly, goal conditions are used to determine when goals have been achieved. A goal can then either be *active* or *inactive*, depending on whether or not its set of "blocking goals" is empty. A goal G is removed from M when at least one of its goal conditions $C \in C_G$ is met. The state of the robot is computed based on the old state using `updateState`, while the default action (i.e., output) is chosen in `defaultAction` based on *R-out*. All goals are then updated in `updateGoal` based on *R-out* and any possible changes detected via sensory inputs *S-change* and the active goals are used to determine the action of the robot via the `chooseAction` function based on the goal and the chosen action so far.

```
FUNCTION updateRADIC(R-out,S-change,D-goals,G-strat)
static M ← ∅
static currentState ← ∅
static oldState ← ∅
M ← M + convertGoal(D-goals, G-strat)
oldState ← currentState
currentState ← updateState(S-change)
action ← defaultAction(R-out)
for all G ∈ M do
  for all C ∈ C_G do
    if C_G then
      M ← M − {G}
    end if
  end for
  G ← updateGoal(R-out, S-change)
  if B_G = ∅ then
    action ← chooseAction(G, action)
  end if
end for
return action
```

Figure 2: The generic update algorithm for RADIC.

Note that the above algorithm is kept as generic as possible, allowing application to the broadest range of deliberative and reactive layers. To apply it to a particular architecture (e.g., to integrate two pre-existing layers), the five functions `convertGoal`, `updateState`, `defaultAction`, `updateGoal`, and `chooseAction` need to be implemented. RADIC can also be used in a standalone fashion with either type of layer, so long as the functionality of the other layer is implemented internally. Reactive layer functionality requires mapping goals to actions, while deliberative layer functionality requires appropriate internal representations (akin to the symbolic layer in SSS).

Unlike other hybrids, RADIC satisfies all of items **I1-4**. Minimal changes to each layer are required for integration (**I1**), as layer output is simply redirected as RADIC input; RADIC relies on internal data structures and functions for its operation. The RADIC component operates independently of either layer, performing its tasks in parallel with the layers' operation, responding to whatever input is received from either layer (**I2**). Computational cost is small (**I3**), as only state updates, goal list updates, and action choice are required. Updates of the goal list require both a measure of state change (via `updateState`) and a subsequent application of the calculated change for each point (via `updateGoal`). Only "active" goals are considered in modifications of the reactive layer's output at any given time. Finally, combining the layers improves overall performance (**I4**) by augmenting the reactive layer with planning capacities and providing the deliberative layer with the ability to act. Furthermore, custom improvements can easily be incorporated into the `updateState` (such as additional processing of *S-change*) and `updateGoal` (such as goal reordering or progress monitoring) functions.

4. CONCLUSION

RADIC has been experimentally validated and compared with other hybrid architectures in a trajectory following task. Beyond demonstrated performance improvements (and unlike other hybrid architectures), RADIC has the benefit of satisfying design desiderata **I1-4**, in the most extreme case actually substituting for a layer. The RADIC algorithm is not limited to navigation tasks, but can be used as a generic action sequencer in hybrid architectures for *any task*. RADIC can be attached to *any reactive layer* by virtue of its method of intercepting reactive output. Moreover, it can subsequently be used with *any deliberative layer* for which the `convertGoal` function is implemented.

5. REFERENCES

[1] J. Albus. 4-D/RCS reference model architecture for unmanned ground vehicles. In *Proc. of ICRA*, 2000.

[2] R. Arkin and T. Balch. AuRA: Principles and practice in review. *JETAI*, 9(2-3):175–189, 1997.

[3] R. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack. Experiences with an architecture for intelligent, reactive agents. *JETAI*, 9(2/3):237–256, Apr 1997.

[4] J. Connell. SSS: A hybrid architecture applied to robot navigation. In *Proc. of ICRA 1992*, pages 2719–2724, 1992.

[5] R. J. Firby. Task networks for controlling continuous processes. In *Artificial Intelligence Planning Systems*, pages 49–54, Chicago, IL, June 1994.

[6] E. Gat. Integrating planning and reacting in a heterogeneous asynchronous architecture for controlling real-world mobile robots. In *AAAI*, pages 809–815, 1992.

[7] R. Jensen and M. Veloso. Interleaving deliberative and reactive planning in dynamic multi-agent domains. In *Proc. of the AAAI Fall Symposium*. AAAI Press, 1998.

[8] K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti. The Saphira architecture: A design for autonomy. *JETAI*, 9(1):215–235, 1997.

[9] D. Lyons and A. Hendriks. Planning as incremental adaptation of a reactive system. *Robotics and Autonomous Systems*, 14(4):255–288, June 1995.

[10] P. Maes. Situated agents can have goals. In P. Maes, editor, *Designing Autonomous Agents*, pages 49–70. MIT Press, 1990.

[11] H. Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(2):205–244, 1995.