GLUE - A Component Connecting Schema-based Reactive to Higher-level Deliberative Layers for Autonomous Agents

James Kramer **Matthias Scheutz**
University of Notre Dame
Notre Dame, IN 46556
e-mail: jkramer3,mscheutz@nd.edu

Abstract

Several problems need to be addressed when integrating reactive and deliberative layers in a hybrid architecture, most importantly the different time scales of operation and (possibly) different data representations of the individual layers. This paper proposes an interface component called GLUE that is designed to mediate between a schema-based reactive and higher-level deliberative layer. We demonstrate the viability of the conceptual architectural design by defining and implementing a schema-based reactive architecture for a ball following task on an autonomous agent, which is then extended by the GLUE component to interface with a simple deliberative layer.

Introduction

Hybrid architectures for autonomous agents are layered architectures intended to combine the benefits of the indivual layers, while eliminating or reducing the effects of their shortcomings. Typically, hybrid architectures combine the functionality of a fast, low-level reactive layer with that of a slow, high-level deliberative layer. The general approach in constructing such hybrid architectures is to add deliberative layers "on top" of a reactive layer, keeping them conceptually separate. There are several reasons for maintaining this separation.

For one, different designs and functional aims underwrite the layout of the two layers. The reactive layer is meant to provide a tight sensor/effector coupling, allowing the agent to react continuously in response to stimulus in a dynamic environment, whereas the deliberative layer is meant to provide higher-level functionality (e.g., planning or reasoning) that allows the agent to do more than simply react to a particular situation.

Secondly, the functional separation of layers implies levels of different computational cost. Computing sensor-motor mappings is generally much less expensive than performing complex "what-if" reasoning or maintaining an updated representation of the environment.

Finally, the layers differ in the extent to which they maintain and keep track of internal and external states. As Gat (Gat 1998) has pointed out, the reactive level keeps state to a minimum, if any is kept at all. Sensory information continuously arrives, is processed and passed on to the effectors. In contrast, higher architectural layers store information and use it to perform complex processing.

A major consideration of the designer of a hybrid architecture is the *integration* of the layers.[1] Potentially, they not only work at different time scales, but also require different processing capacities and resources, and may even use different data formats and structures at the interfaces of their components. In particular, in the domain of autonomous agents, where fast-working, highly responsive reactive layers have proven necessary (e.g., (Jensen & Veloso 1998; Maes 1990)), the integration of a slow deliberative layer, which can interact with and sometimes take control of the reactive layer, has been a challenge.

In this paper, we propose the use of a component between the reactive and deliberative layers that not only simplifies the integration of layers, but also improves reactivity. The next section details some methods of integration previously implemented, including a description of their relative strengths and weaknesses. Then issues of integrating deliberative functionality with a schema-based reactive layer are discussed and a generic link unit for (architecture) extensions is proposed that permits deliberative extensions to schema-based reactive layers with only a minor modification to the existing reactive layer. Finally, we demonstrate the GLUE component by first defining a schema-based reactive architecture for a ball-following task, and then augmenting it with a simple deliberative extension using the GLUE component. The architecture has been successfully implemented on an autonomous robot, thus verifying the viability of the conceptual design.

[1] Note that by "integration" we do not intend what could be called the "Omega model", where the layers have a sequential processing function: sensory information comes in via low level sensors, gets abstracted as it goes up through higher central layers, until action options are proposed near the top, where some decision is taken, and control information flows down through the layers and out to the motors (e.g., see (Nilsson 1998),(Albus 1981)). Instead, we are referring to an arrangement of layers, where each layer keeps performing its functions on its own time scale, while being able to interact with other layers.

Hybrid Architectures

There have been several methodologies used to integrate reactive and deliberative layers in hybrid architectures for autonomous agents. Each has its own particular strengths and weaknesses, and none can be said to be overwhelmingly the best. In fact, it has been noted (Nwana 1995) that hybrid architectures generally suffer from at least three high-level problems:

- ad hoc or unprincipled designs that not only require hand crafting of each control module, but may also overlap functionality
- application-specific tasks, which may very well be unavoidable
- unspecified supporting theory

Furthermore, there are also practical difficulties with representations between layers, error-detection across layers, and timing issues related to the different time scales on which the layers operate.

We will briefly review two hybrid architectures that are of special interest here: the first is "Atlantis", because it maintains the conceptual separation and functional autonomy of multiple layers while trying to connect them, and "AuRA", because it attempts to combine a schema-based reactive layer with non-schema-based deliberative extensions.

Atlantis

Gat (Gat 1998) identified and made explicit the role of state in separating architectural layers, which was a much-needed explication of conceptual division. "Atlantis" is a clear example of a three-layer architecture. The reactive layer (the "controller") is composed of feedback control loops that maintain little or no state. These control loops are handcrafted and given the operational name of "behaviors". The second layer is the "sequencer", which serves to determine a behavior appropriate to the current environmental situation by maintaining a history of previous states. (Gat 1998) notes that "the sequencer should not perform computations that take a long time relative to the rate of environmental change at the level of abstraction presented by the controller". The third layer is the "deliberator", implemented as a query/response system. In other words, the sequencer requests a plan from the deliberator that is then put into action.

While "represented state" as a criterion for the separation of layers can be useful, it is not in and of itself sufficient to suggest design options for linking the layers. Yet, the linkage is a necessary step in defining an interface between layers (generally from higher to lower), allowing information from one to be used by the other. This is acknowledged by Gat when he points out that a major difference between Atlantis and other architectures is the fact that different representations can be (and are) used in different layers of the Atlantis architecture. While the ability to use a variety of representations can be a major benefit in the design of an agent's architecture, the issue has not been explicitly addressed. Instead, it is generally only considered as part of the implementation.

AuRA

Arkin's (Arkin & Balch 1997) "Autonomous Robotic Architecture" (AuRA) is an example of a schema-based architecture. The reactive layer uses two classes of schemas, a perceptual and a motor schema, and is generally structured as shown in Figure 1.

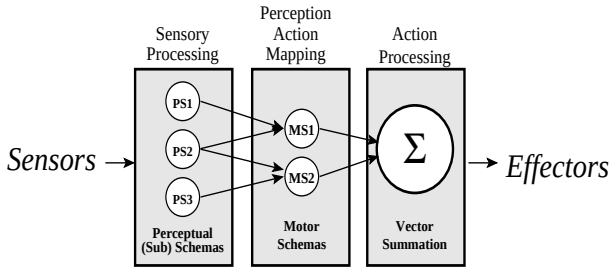

Figure 1: Basic schema-based reactive architecture

Motor schemas take as input the output from possibly several perceptual schemas (which are themselves connected to environmental sensors) and combine it to implement a particular motor behavior. Output from different motor schemas is then "summed up" and passed on to the effectors.

Schemas have many benefits, including:

- coarse grain granularity (for expressing the relationships between motor control and perception)
- concurrent actions (in a cooperative yet competing manner)
- inexpensive computations (as only the current field need be calculated)
- behavioral primitives (from which more complex behaviors can be constructed)
- biological plausibility

However, the basic integration principle in Aura is to turn on or off whole sets of motor behaviors, engaging the reactive layer and halting deliberative execution. Behavior selection is based on "error detection": the deliberative layer starts by determining a set of perceptual and motor schemas to activate, which then take over and remain in operation until "failure" is detected (such as lack of progress towards the goal as indicated by zero velocity or a time-out). At that point, the planner is reinvoked one stage at a time, and the cycle of determining an appropriate set of behaviors, running them, etc. repeats itself.

Issues with Integrating Higher Level Components into Schema-Based Architectures

Orignially, AuRA used a rule-based system to control behavior selection. This was replaced by a finite state machine implementation as the means of plan execution and control. Both solutions work by engaging or disengaging particular behaviors (i.e., sets of motor schemas) based on perceptual input. For instance, the `escape` behavior, defined as "evade

intelligent predators", will not be activated unless and until an "intelligent predator" is sensed. The solution of switching on or off whole sets of behaviors is a fairly general way of connecting higher layers to a lower-level layer. It can be successfully applied to motor schemas in a schema-based reactive layer in the AuRA architecture, as well as for controllers in the reactive layer in the Atlantis architecture. Yet, the fact that the deliberative extensions use whole sets of motor schemas or controllers also shows that such deliberative extensions are not well integrated in the system, for if they were truly integrated, they would not have to reconfigure the reactive layer in such a holistic manner. As a consequence, such hybrid architectures do not really address the kind of interplay between deliberative and reactive control that has been part of the motivation for looking at hybrid architectures in the first place (e.g., the integration of a deliberative planner with a reactive plan execution mechanism). In the case of a deliberative planner, for example, the worst case scenario may require different sets of schemas (i.e., configurations and combinations of motor schemas) for every single plan step, which, in turn, will almost completely remove the autonomy of the lower level (besides the computational overhead of maintaining all the different configurations of motor schemas). Or to put it differently: lower reactive layers "degenerate" to mere implementations of higher level "actions" (similar to action sequencing performed in contention scheduling (Norman & Shallice 1980; Cooper & Shallice 2000)).

In order to retain in part the autonomy of lower levels and to minimize the computational overhead and modifications of the lower reactive layers required for an integration of higher deliberative layers, we investigate an alternative option of integrating schema-based reactive systems with deliberative extensions: *the modification of outputs from perceptual schemas*.

Inputs to motor schemas are usually outputs from perceptual schemas, which in turn are connected to environmental sensors. These inputs have a particular representational format in schema-based architectures: they are *force vectors*. So long as the inputs to the motor schemas take the form of force vectors, they need not originate via the environmental sensors, but can be fabricated through other means.

There are two requirements for input that does not come from the environmental sensors: (1) data transformation – it needs to be transformed into a force vector representation, and (2) data maintenance – it needs to be provided and maintained at a time frame appropriate to the reactive layer.

Data Transformation

The deliberative layer generally, although not necessarily, uses a more abstract representation of sensory data than the reactive layer. This allows a variety of existing AI techniques to be used. However, when a different representation is used, it must be transformed into data the reactive layer can use which, in the case of schema-based reactive layers, is a vector representation. Once data transformation is performed, the vector can be passed as input to the motor schema, where it will be treated as if it had originated from one of the perceptual schemas.

Data Maintenance

If the deliberative layer operates on the same time frame as the reactive layer, its output can be processed along with that arriving from the environmental sensors. However, as noted earlier, deliberative layers are generally slower than reactive layers, which raises the problem of maintaining the representation sent to the reactive layer and adjusting it to environmental changes. As the agent performs actions on the environment, information originating in the deliberative layer may become increasingly obsolete as the time differences between deliberative and reactive processing increases. Reducing this source of error is a major problem that must be addressed by any kind of integration mechanism.

There seem to be at least two approaches to containing time lags. The first is to allow the representation from the deliberative layer to persist only for a limited time, contingent on both the operational time scale of the deliberative layer and the rate of environmental change (which is in turn dependent on both the actions of the agent and external factors). The idea is to update information *often enough* to ensure that the error does not affect the operation of the agent.

The second approach attempts to minimize the error in the deliberative layer's representations by updating them based on feedback from the effectors. This requires the specification of how an agent's actions affect its representation of the environment. This approach may increase the amount of time the deliberative layer has to perform its processing *enough*, allowing it to update the representation. For instance, if an agent's action is to move forward for a given distance, the stored state would be updated as if it were sensed in the new relative position. However, this only improves the first approach; it does not eliminate the discrepancy between the environment and the internal representation, which will continue to grow, albeit at a reduced rate.

GLUE - A Generic Link Unit for (Architecture) Extensions

From the analysis above, we derive a functional specification for the GLUE component that can mediate between deliberative and reactive layers. Inserted as an interface between layers, it could be used at any level in the architecture. Here, however, we will focus on the interface between the schema-based reactive layer and a higher level deliberative layer. To provide a means of interaction between these layers, so-called *sticky points* are isolated, i.e., places where the GLUE component is attached to the respective layers. On the reactive layer side, the point of attachment is a motor schema, on the deliberative side, it can be any "perceptual output" of the deliberative layer (see below). Note that it may be necessary, depending on the desired functionality, to add separate sticky points to each individual motor schema.

Functional Specification of GLUE

As mentioned above, data from the deliberative layer must be in a form the reactive layer can use, and then be maintained. To meet both of these requirements, the GLUE component must have two forms of input: the deliberative layer's output and the effector feedback. The deliberative layer's

output is transformed into a force vector, which is then added to the data from the environmental sensors and updated with feedback from the effectors each time new perceptual data is available. This cycle is repeated until the deliberative layer produces new information, at which point the "maintained" data is discarded (see Figure 2).

An outline of the data flow progression is:

1. receive data from the deliberative layer
2. transform that data for the reactive layer
3. output transformed data to the reactive layer, adding it to data from the environmental sensors
4. update the transformed data using the effector feedback
5. repeat from step 3 until new information is available from the deliberative layer
6. once new information is available from the deliberative layer, repeat from the beginning

The output of the GLUE component can have a variety of purposes and effects. It can cancel out perceptual information, effectively nullifying or altering environmental sensors' effect on the motor schemas. Alternatively, it can also add perceptual information that is not actually present in the environment.

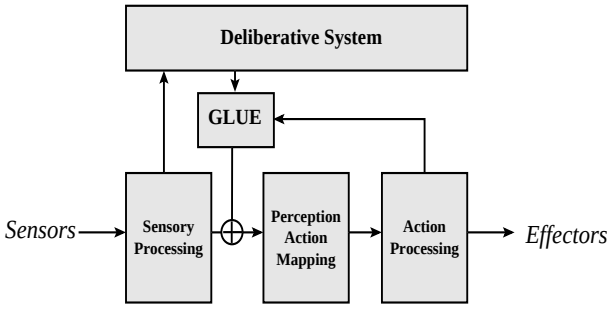

Figure 2: Architecture with GLUE component

An Application of the GLUE Component in an Autonomous Robot

To verify the conceptual design of the GLUE component and test its feasibility and viability on a real-world agent, we devised a schema-based architecture for an object tracking and following task. In this task, a robotic agent has to avoid obstacles in its environment and move towards an object of a particular color whenever it happens to spot one. The object at hand was an orange soccer ball, which the robot could easily identify by its color.

In a first step, we defined a schema-based architecture for this task, which implements obstacle avoidance and object tracking and following behavior. In a second step, we augmented it by the GLUE component and a simple deliberative extension, which adds a "ball dribbling" behavior to the existing behavioral repertoire.

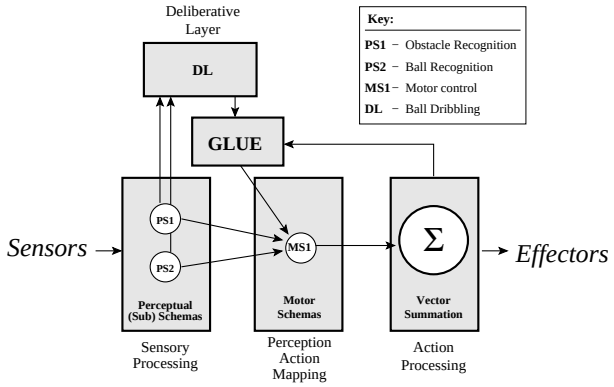

Figure 3: Integration of schema-based reactive layer and simple deliberative layer using the GLUE component

The Reactive Layer

Two basic behaviors have been defined: avoid_obstacle, which uses sonar to identify objects near the robot and avoid collisions, and follow_ball, which uses the camera to identify the ball based on its color. The avoid_obstacle acts as a repulsive force in the operation of the robot, while the follow_ball acts as an attractive force. The combined behaviors make the robot approach the ball whenver it happens to see it, while avoiding obstacles (note that it may lose track of the ball in the process of getting around obstacles). At a certain distance from the ball, the two behaviors will cancel out; the attraction from the follow_ball behavior will be cancelled out that of avoid_obstacle. We purposely did not implement additional behaviors, such as one that would locate the ball or identify other players, in order to provide an elementary base case.

The Deliberative Layer

The deliberative layer uses information from the environmental sensors to determine a plan. As a proof of concept, the generated "plan" is to continue following the ball, overriding the fact that it is perceived as an obstacle and acts as a repulsive force. This results in a "dribbling" behavior – the robot bumps into the ball, forcing it ahead, then proceeds to follow it again. The same technique can be employed in the context of robotic soccer, where color is used to identify teammates, opponents, and field markers. A more complete deliberative system would use the additional information to determine an appropriate "play", such as preparing to receive a pass or defending the goal.

GLUEing the Layers Together

The GLUE component was then added to the functioning schema-based reactive layer as a means of providing an interface with the deliberative layer. For the test under consideration, this entailed using the information from the environmental sensors to make the ball more attractive. An additional force vector was produced to cancel out the repulsion of the avoid_obstacle behavior produced by the

ball. The feedback from the effectors consists of a velocity reading and the time elapsed since the last update, which is used to determine how far the robot had travelled. The GLUE component operates asynchronously from the integrated layers, generally at a time frame between that of the layers. The fabricated force vector was then updated, reflecting the robot's actions.

Experimental Setup and Results

The architecture was implemented on an a Pioneer P2-DXe mobile robot from ActivMedia using the AgeS agent development environment (under development in our laboratory). The robot gathers information about objects in the environment using a ring of sixteen sonar sensors for distance and angle information and a camera with supporting software for color and blob detection. Every object that does not have the color of the target object is considered an obstacle. The environment is a square area of about 20 by 20 feet, containing one ball of the target color and a few obstacles of other colors. All processing was performed on board, making the robot completely autonomous.

When only engaging the reactive layer, the robot displays two types of behavior, dependent on whether the ball is in motion or not. If a moving ball enters the robot's field of vision, the robot will follow it while avoiding obstacles. If a stationary ball is within the robot's field of vision, it will move to the vicinity of the ball and then sit in front of it. With the deliberative layer GLUEed onto the reactive layer, the robot proceeded to exhibit dribbling behavior.

Figure 4: The robot used in the experiment equipped with a kicking mechanism used for ball dribbling

Discussion

GLUE components provide a methodology for integrating layers in a hybrid architecture, useful both in conceptual design and actual implementation. Conceptually, they preserve the functional separation between layers by providing a designated place to interface layers and necessitating the clear definition of the data transformation. In implementation, GLUE has additional practical benefits. First, it is non-intrusive; by adhering to the conceptual division GLUE enforces, integration between layers makes use of already existing structures. Further, since the use of those pre-existing structures requires little or no modification to the structures themselves, the operation of previously tested components will not be altered. Second, the GLUE component should be usable in a variety of systems, so long as it is possible to transform data between layers. Lastly, by providing a means of data persistence and maintenance, it is possible for layers to operate in different time frames. The need for such a mechanism becomes apparent when considering that the deliberative layer is typically much more computationally intensive than reactive mechanisms and therefore requires more time to produce useful results. Allowing data from a "slow" layer to persist in a form usable by a "fast" layer increases the interplay between the two. This provides a more coherent integration between layers than turning entire behaviors on or off.

To prove the viability of GLUEing layers together, first a simple schema-based, reactive system was implemented which followed a ball while avoiding obstacles. Then, a simple deliberative task was chosen – use the identification of the ball to override its status as an obstacle. Although this functionality could be built directly into the reactive layer, it served as an appropriate test case for a GLUE component. Future work will expand the number of behaviors available in the reactive layer, in addition to extending the functionality of the deliberative layer to be able to test the GLUE component in a more complex agent.

References

Albus, J. 1981. *Brains, Behaviour and Robotics*. Peterborough, N.H.: Byte Books, McGraw Hill.

Arkin, R. C., and Balch, T. R. 1997. Aura: principles and practice in review. *JETAI* 9(2-3):175–189.

Cooper, R., and Shallice, T. 2000. Contention scheduling and the control of routine activities. *Cognitive Neuropsychology* 17(4):297–338.

Gat, E. 1998. On three layer architectures. In Kortenkamp, D.; Bonnasso, R. P.; and Murphey, R., eds., *Artificial Intelligence and Mobile Robots*. AAAI Press.

Jensen, R., and Veloso, M. 1998. Interleaving deliberative and reactive planning in dynamic multi-agent domains. In *Proceedings of the AAAI Fall Symposium on on Integrated Planning for Autonomous Agent Architectures*. AAAI Press.

Maes, P. 1990. Situated agents can have goals. In Maes, P., ed., *Designing Autonomous Agents*, 49–70. MIT Press.

Nilsson, N. 1998. *Artificial Intelligence: A New Synthesis*. San Francisco: Morgan Kaufmann.

Norman, D., and Shallice, T. 1980. Attention to action: Willed and automatic control of behaviour. Technical report, University of California, San Diego, CA.

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