Appendix A: System Architecture

Figure 20, on page 141, summarizes the architecture of the two testbeds built for this research. In general, the learning system is connected “at arm’s length” to the microworld. The learning system can only send simple commands (one of a small number of actions) to the microworld, and it only receives what sensor information the microworld transmits back. It does not have access to internal microworld state.

The learning system is the same in either scenario. As shown in Figure 20, it contains the schema system proper (which consists of, essentially, a reimplementation of Drescher’s schema system [Drescher 91] without certain elements),[1] plus the goal-independent focus mechanism described in Chapter 3 and the goal-dependent mechanism described in Chapter 4. The action selection system picks actions at random in both the unfocused reimplementation of the original algorithm and in the goal-independent work described in Chapter 3, but is informed by the goal system in the further work described in Chapter 4. In addition, the learning system contains a large amount of diagnostic and performance-monitoring code, from which the results (in terms of work per schema, etc) described in this research were derived.

The microworld used in each scenario is, of course, different. In the case of the infant/eyehand scenario, it runs in the same process as the learning system, though its only connections to the learning system are via the aforementioned sets of actions and sensory bits. In this case, the entire system was implemented as a single process in Lisp under Symbolics Genera.[2] The microworld itself has no sophisticated rendering apparatus; instead,

1. Such as the composite-action system or the mechanism for computing delegated or instrumental value.

simple character-based diagrams can be produced of its current state for inspection, evaluation, and debugging.[3]

In the Hamsterdam scenario, however, the microworld is a separate process, implemented as C and C++ code using SGI Inventor, and runs on a Silicon Graphics workstation. Hamsterdam includes sophisticated, real-time, three-dimensional graphic rendering. It communicates with the learning system (in this case, implemented in Harlequin LispWorks and running in a separate process, on either the same machine as Hamsterdam or a different one) via a network connection consisting of a UNIX TCP socket.[4] This proved to be implementationally less than ideal, but of no interest theoretically, because the already-enforced "arm's length" relationship between the learning system and any associated microworld already decreases the possible coupling between the learning system and its microworld to a very loose connection.

In both cases, the Lisp portion of the system (learning system and microworld in the infant/eyehand scenario, or the learning system only[5] in the Hamsterdam scenario) could be snapshotted to disk and restored later using a component called the *snapshotter*. This was an implementation convenience to make it easy to duplicate runs with different param-

2. Since the system is coded in Common Lisp, it could run under any Common Lisp implementation, however. It currently runs under Symbolics Genera and Harlequin LispWorks, and ports to other lisp implementations would be extremely straightforward.

3. A simple, colorful, graphical representation of this world was also constructed to demonstrate certain aspects of the early system, but was essentially equivalent to the character-based diagrams in content.

4. Were LispWorks able to directly incorporate the code and libraries of Hamsterdam, the learning system and Hamsterdam could run in a single process. However, because they were required to run in separate processes due to limitations in currently-released versions of LispWorks, there is no reason why any learning system, running on any other machine, could not be substituted for the current configuration.

5. The state of the Hamsterdam microworld cannot be so preserved in the same fashion, having never been designed for it. Thus, strict reproducibility of runs in the Hamsterdam scenario is not possible, due to the differing environments that would be faced even by "identical" runs of the learning system. This makes the Hamsterdam scenario slightly more difficult for debugging, since events would not always unfold identically even if the same code were to be run.

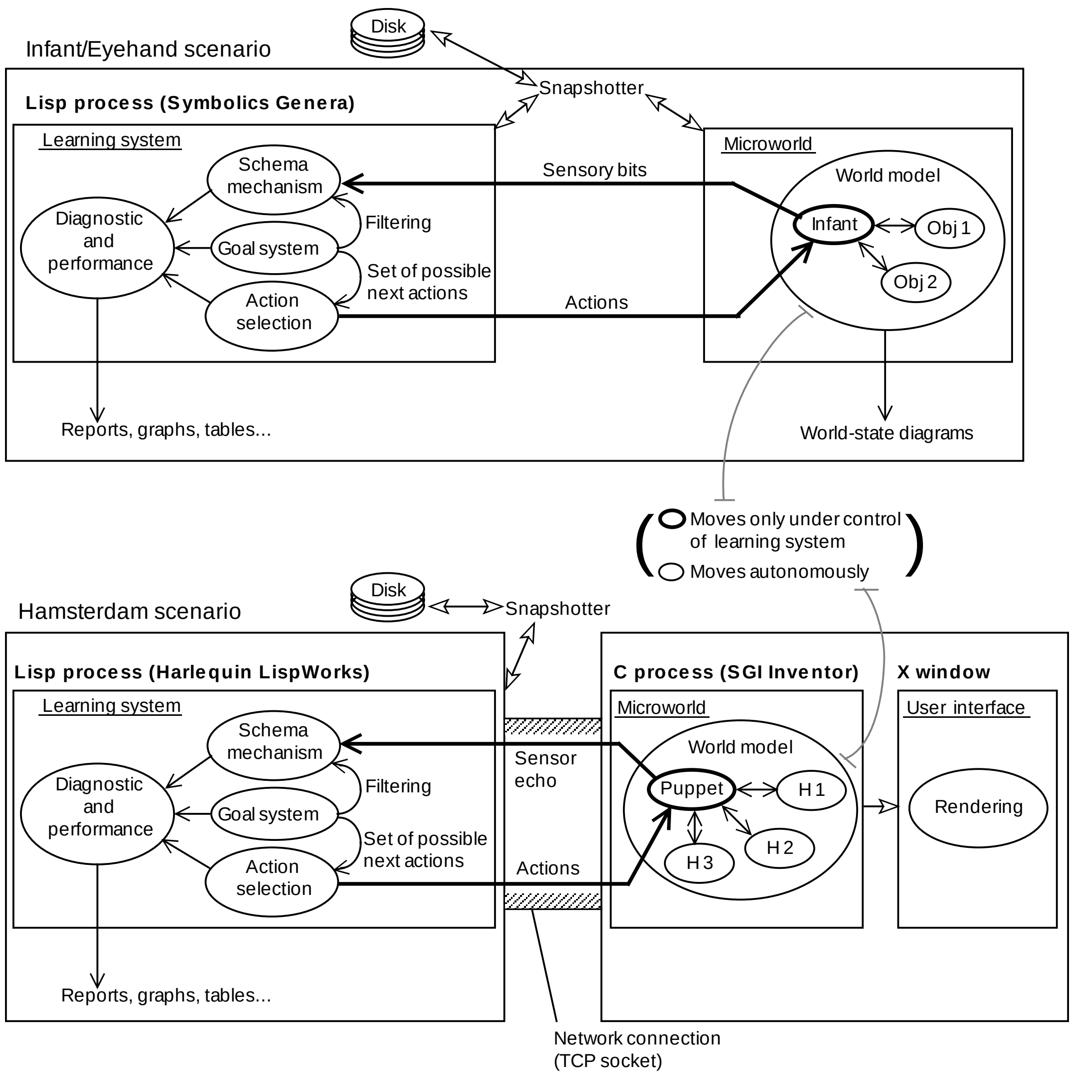


Figure 20: System architecture of both scenarios

In the infant (top) scenario, everything runs in a single process. The only object under control of the learning system is the infant; the other objects in the world occasionally move by themselves.

In the Hamsterdam (bottom) scenario, the learning system runs in one process, and Hamsterdam runs in a separate process. The only object under control of the learning system is the puppet; the hamsters and predators are free to wander around autonomously, and do so continuously.

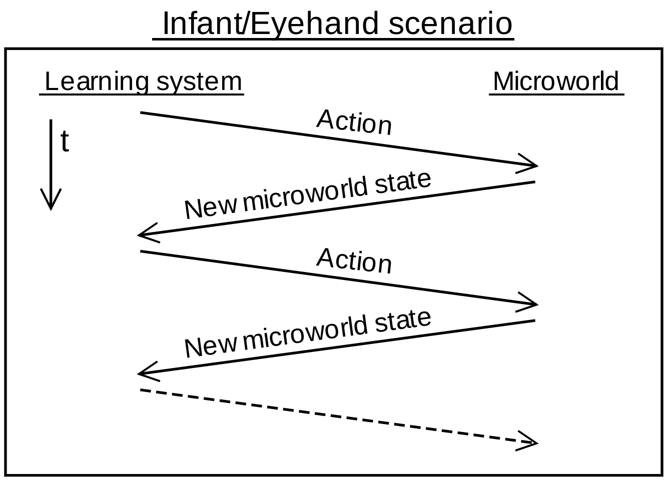
eters, or to return to a long run after a reboot, given that the unmodified, unfocused system could take up to 3-4 days to produce 3000 to 4000 schemas.[6]

A communications signature of the interactions between the learning system and any given microworld is summarized in Figure 21, on page 143. It is expressed in the typical fashion for network protocols, as a Feynman diagram. In general, there is a lockstep relationship between an action being requested by the learning system, and the corresponding sensory bits being returned by the microworld after the action has been performed. This lockstep relationship, while perfectly reasonable in the infant/eyehand scenario, is less realistic in a more dynamic microworld, such as that employed in the Hamsterdam scenario.

Because of this difference in microworld characteristics, the communications protocol employed for the Hamsterdam scenario was slightly modified. An explicit *null action,* not present in the infant/eyehand scenario, was introduced. Like any other null action, it is able to be part of a generated schema. Strictly speaking, such a null action could simply be modelled as an action that never produces any detectable result, except for taking some amount of time (and, indeed, this is exactly how it was implemented). However, it serves as a placeholder in the schema system to model the *result of not doing anything*, which is itself an important concept in a dynamic world.[7]

6. While the snapshotter's contribution was considerable early on in the course of this research, its utility steadily decreased, because each successive refinement to the learning system, as described in later chapters, served to increase the speed of the resulting system and thereby decrease both the real time and the computational work of producing equivalent states. In some cases, when using the most highly-selective learning, runs of useful size could be produced in an hour or two—a considerable improvement.

7. Consider a robot which must outwit a motion detector. If it never stops and waits, it will never have a chance to observe the little red light on the motion detector go out. This means it will never be able to correlate its motion in the environment with the behavior of the motion detector, since the detector will always be detecting movement, and the light would always be on.



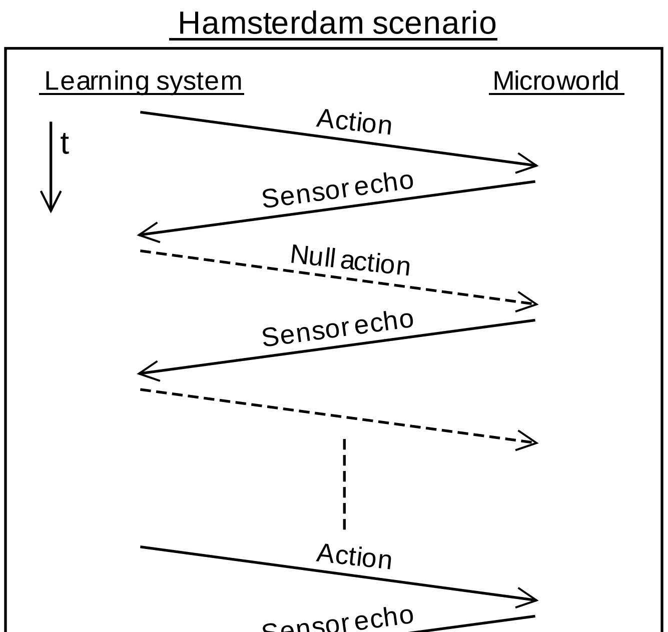


Figure 21: Communication signature for both scenarios

Time flows downward in both diagrams. In both scenarios, the learning system requests an action, has it performed in the microworld, and gets a set of sensory bits returned. In the Hamsterdam case, an explicit null action is also possible.

By explicitly representing null actions, we allow changing the ratio of the number of “real” actions performed to the number of “null” actions performed. Let us define this ratio as follows:

$$\alpha = \frac{A}{\varnothing A}$$

where A is the number of “real” actions taken over some interval, $\varnothing A$ is the number of “null” actions taken over the same interval, and α is therefore a measure of behavior: if α is substantially *less* than unity, the behavior of the system can be said to be *shy* or *inhibited*, whereas if α is substantially *greater* than unity, the system’s behavior can be said to be *outgoing* or *audacious,* to use some rather anthropomorphic terms. The parameter α thus behaves vaguely like the level of limbic-system activation in the mammalian brain. When the learning system is outgoing, taking action most of the time and never pausing to watch the world go by, it can learn a great deal about the effects of individual actions it takes, but not how the world functions if it were not taking actions at all. Conversely, when the learning system is shy, not taking action and instead observing how the state of the world changes when it does nothing, it can learn how doing nothing affects the world.[8]

There is a secondary reason why null actions were introduced. The Hamsterdam microworld is continually updating its internal state, because its agents operate in real time, and it must continue to update so as to re-render the scene and preserve the illusion of a dynamic, changing world. Such updating happens several times a second, which is substantially faster than the learning system can keep up with the world on the current hard-

8. If the system were provided with a much more sophisticated source of data about the world, which corresponded to being told about the actions taken by other agents in the world, it would be possible for it to learn the correlations between other agents’ actions and their effects. However, no source of data like this exists in the current scenarios studied; to make such a source available, the control structure of the learning system would have to be modified to substitute other agents’ actions for its own while computing statistics and producing schemas. Presumably, the right time to make such a substitution would be when the agent is otherwise performing a null action, or some action whose result is “overlearned” and hence whose results are no longer interesting to the learning system.

ware, even with the improvements described in later chapters. Were the learning system to accept every update, it would be forced to discard many of them, or it would lag further and further behind the current state of the world. Yet, the microworld cannot be expected to know how fast the learning system can run. (Any such attempt would be doomed to failure, given that the learning system's running speed varies dramatically based on what sort of focus it employs and how much it already knows, and because both the learning system and Hamsterdam are not tied to particular platforms.)[9]

Null actions provide an explicit solution to this dilemma. By allowing the learning system to dictate exactly when it receives sensory input (it always gets one update after any action, whether real or null, no more and no less), the learning system polls the microworld for sensor updates. Therefore, the microworld's sensor update rate is throttled by the learning system. This is reasonable, since more-frequent updates would be useless to the learning system anyway. If the learning system is quite slow in requesting updates compared to the speed at which the microworld runs, it will miss many important events and may in fact not learn anything useful. Thus, any real agent employing this approach would have to be placed in a situation where its cognitive speeds are up to the task of the world with which it must interact, a familiar problem in both engineering and biology, or would at least require the ability to be infrequently but quickly interrupted if some high-priority sensory signal (such as being about to go over a cliff) demanded prompt attention.

9. The learning system can run under Symbolics Genera, at a variety of speeds depending upon the type of Lisp Machine in use, or under Harlequin LispWorks, again at a variety of speeds depending on the type of SGI used. Hamsterdam can run on many kinds of SGI platforms, each at a different speed.