Concurrent Knowledge Activation Calculation in Large Declarative Memories

Scott A. Douglass (scott.douglass@mesa.afmc.af.mil)
Christopher W. Myers (christopher.myers2@wpafb.af.mil)
Air Force Research Laboratory, 6030 S. Kent St.
Mesa, AZ 85206 USA

Abstract

To behave effectively and flexibly in complex situations, models specified in cognitive architectures must be able to store and access large amounts of declarative knowledge. However, as research efforts employing cognitive modeling grow in scope and complexity, currently available modeling tools, languages and cognitive architectures are being pushed to their practical limits. This paper describes research looking specifically at how a large declarative memories challenge the current implementation of ACT-R and describes an applied effort to develop an alternative implementation of ACT-R's retrieval process. The alternative exploits concurrency features of the Erlang programming language to extend the practicality of ACT-R's retrieval mechanisms to new levels of scale. The ideas and methods underlying the alternative implementation are general and illustrate how concurrency can accelerate calculation in other architectures struggling to support large associative declarative memories.

Keywords: declarative memory; concurrent activation calculation; semantic networks; ACT-R; Erlang.

Introduction

As research efforts employing cognitive modeling grow in scope and complexity, available modeling tools and languages are being pushed to their practical limits. For example, the implementation of ACT-R within the Lisp programming language may hinder the development of large-scale models due to limitations in declarative storage capacities (Douglass, 2009). If cognitive modeling is to grow in scope and complexity, we must meet the challenges underlying these limits.

An AFRL large-scale cognitive modeling (LSCM) initiative is currently exploring potential solutions to these challenges. The LSCM initiative is committed to integrating well understood mechanisms from cognitive architecture research into new modeling approaches that facilitate model scaling. For example, the empirical strength of ACT-R's declarative system (Anderson, 2007) has motivated us to ensure that the LSCM initiative's solutions preserve ACT-R's declarative memory mechanisms.

LSCM initiative efforts to develop domain-specific modeling languages (DSML-s) supporting increased model scale and persistence involve efforts to increase the scale and persistence of a declarative memory system that mimics ACT-R's. This paper describes recent efforts to retain and scale ACT-R's memory mechanisms in a modeling and simulation framework supporting RML1 (research modeling language), the first DSML developed in the LSCM initiative. RML1 is a generic DSML tailored to the needs of cognitive modeling. RML1 has a hybrid (graphical and textual) syntax, and executes in a runtime environment implemented in the Erlang programming language (Armstrong, 2007).

Modeling with Large Declarative Memories

In the following sections we provide a brief overview of ACT-R and describe how to extend ACT-R's declarative retrieval process by "carving it up at the joints." We conclude this section with a discussion of replicating top-down (i.e., endogenous) and bottom-up (i.e., exogenous) constraints on ACT-R's memory retrieval process.

Brief Overview of ACT-R

ACT-R is a cognitive architecture for developing computational cognitive process models (Anderson, 2007). In ACT-R, cognition revolves around the interaction between a central production system and several modules. There are modules for vision, motor capabilities, memory, storing the model's intentions for completing the task (i.e., the control state), information retrieved from memory, and a module for storing the mental representation of the task at hand (i.e., the problem state). Each module contains one or more buffers that can store one piece of information, or chunk, at a time. Modules are capable of massively parallel computation to obtain chunks. For example, the memory module can retrieve a single chunk from long-term memory and place it into the module's buffer.

Chunks are defined by the modeler to have a particular type, or chunk-type, and a set of key-value pairs. Retrieval in ACT-R is based on a combination of: (1) endogenous influences expressed in retrieval constraints; and (2) exogenous influences originating from chunks in the slots of buffers assigned activation weights by the modeler. When retrieving a chunk, the modeler must specify the type of chunk to retrieve, and all chunks of that chunk-type are candidates for retrieval. All candidates' activations are computed, and the one with the highest activation is retrieved. Chunk activation can be exogenously influenced (i.e., primed) by spreading activation from other modules—any module that contains a chunk as the value in a key-value pair spreads activation to related chunks. As the number of chunks in declarative memory increases, the number of candidates during retrieval also increases. As retrieval candidates increase, retrievals may become slow, and in some instances too slow to support large-scale models that must interact with other system components in real-time.

Increasing Scale by Externalizing Chunk Storage

Our initial efforts to extend the viability of ACT-R's retrieval system to large-scale modeling contexts focused on the storage of chunks outside of ACT-R and Lisp. Database management systems (DBMS) such as PostgreSQL can be effectively used to store a large and persistent set of ACT-R declarative memories (Douglass, et al, 2009). This research determined that services provided by the PostgreSQL DBMS can be integrated into ACT-R via a custom "persistent-DM" module. We found that the persistent-DM module greatly reduced ACT-R's storage burden and significantly increased the practical size of declarative memory sets that could be accessed by cognitive models.

The effectiveness of the persistent-DM module was based on the fact that ACT-R's application of retrieval constraints mimics the behavior of a DBMS executing a SQL query. When the persistent-DM module is employed, requests for instances of a particular chunk-type possessing specific sets of key-value properties are translated into SQL queries and then executed to recover matching chunk instances from an external database. "Outsourcing" the storage and recovery of matching chunks through SQL queries in this way is beneficial because of the capacity of PostgreSQL databases and the effectiveness of indexing in relational databases. Unfortunately, while persistent-DM assumed some of the retrieval burden by efficiently isolating the subset of chunks that had to have their activations re-calculated, the module simply relayed them to ACT-R's default serial activation calculation mechanism.

Carving the Retrieval Process at the Joints

We started the development of RML1's memory system by asking ourselves three questions:

Q1. How do the equations that explain activation and associative strengths in ACT-R define the fundamental nature of the ACT-R retrieval process?
Q2. How does the current ACT-R implementation computationally realize the retrieval process?
Q3. Can the fundamentals of the retrieval process be computationally realized in other ways?

Q1 Human memory is more than an information storage and retrieval system. Likewise, declarative memory in ACT-R is more than just a mechanistic account of information storage and retrieval (Anderson, 2007). Human memory is a part of a system that learns and acts in the world. Human behavior is as flexible as it is because we know lots of things and can use what we know to craft contextually appropriate and effective actions in many different circumstances. It is not enough to know a lot; we also have to be able to quickly cull through all that we know in order to retrieve and apply the right knowledge given our circumstances. The crown jewels of ACT-R's memory system are a set of equations explaining how sub-symbolic calculation, learning, and the utilization of activations and associative strengths enable these critical properties of human memory (see Anderson, et al., 2004 and Anderson, 2007 for detailed descriptions). The equations are presented in Table 1 below so that their details—specifically their indexing of chunks *i* and *j*—can be used to confirm a claim that they describe how sub-symbolic properties related to the activations and associative strengths of *individual chunks* influence the probabilities and time costs of their retrievals. That is, the equations precisely explain how activation is calculated for individual chunks in what can be considered independent calculations.

Table 1: Equations describing chunk activation. The key components of the equations are a single focal chunk indexed as *i* and chunks in context indexed as *j*.

Common Name	**Equation**
Activation	$A_i = B_i + \sum_{j \in C} W_j S_{ji}$
Base-Level Learning	$B_i = \ln\left(\sum_{k=1}^{n} t_k{}^{-d}\right)$
Attention Weighting	$W_j = {}^{W}/_{n}$
Associative Strength	$S_{ji} = \ln\big(prob(i\|j)/prob(i)\big)$
Retrieval Time	$Time = Fe^{-A_i}$
Retrieval Probability	$Prob = 1/\big(1 + e^{-(A_i - t)/s}\big)$

Any declarative memory system adhering to ACT-R's theory of human associative memory must minimally calculate each chunk's activation according to these equations. The equations define a fundamental unit of computation scoped around each chunk in declarative memory and abstract away from how the process of retrieval executes all the chunk activation calculations underlying a single retrieval.

Q2 The current ACT-R implementation (ACT-R 6) sequentially realizes all the chunk activation calculations underlying a single retrieval. Hence, chunk activation calculations occur one after the other as a process, not described in the equations above, searches for and retrieves the chunk with the highest activation. To ensure that this point is clear, the retrieval process in ACT-R will now be summarized.

Retrieval in ACT-R is influenced by bottom-up contextual cues and the application of top-down constraints. Retrievals based on top-down constraints generally proceeds in the following way. An "ISA" property in a ***retrieval request*** is used to isolate type-compatible chunks in declarative memory into a ***candidate set***. Slot value constraints representing additional properties required of a chunk contained in retrieval requests are then used to further reduce the candidate set. The activations of chunks surviving all these top-down constraints are then computed in accordance with the equations above. The chunk meeting all top-down retrieval constraints with the highest activation is returned in the retrieval buffer.

The impact of the serial calculation of activation is illustrated in Figure 1 below. The top and bottom diagrams in the figure represent two extreme situations. When activation calculations are computed sequentially, the total time cost is roughly equivalent to a per-activation computation time, *t*, multiplied by the number of chunks. When activation calculations are computed concurrently, the total time cost will be slightly more than *t*. Given that the ACT-R activation equations function in the scope of single chunks and in so doing "modularize" the calculation of chunk activations, we argue that the challenge to extend the scale of ACT-R's memory system is really a challenge to maximize the concurrency of chunk activation calculation during retrieval events.

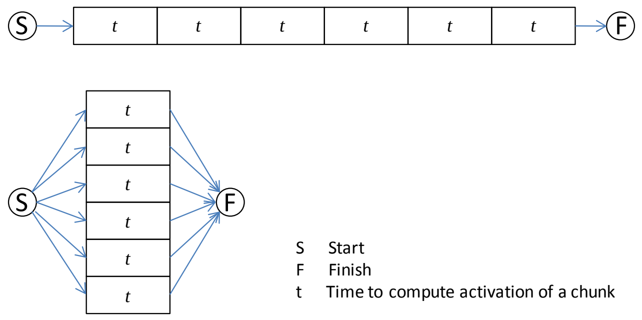

Figure 1: Costs of serial & concurrent activation calculation.

Q3 To find a way to incorporate concurrent activation calculation into the persistent-DM module, we set out to: (1) extend persistent-DM with the ability to compute activations; and (2) develop ways of partitioning databases across multiple PostgreSQL DBMS instances. The first of these challenges was low-hanging fruit; queries to an extended persistent-DM can now include a query capturing top-down retrieval constraints and a representation of context capturing bottom-up sources of activation. Retrievals executed by this version of persistent-DM isolate a sub-set of chunks meeting the top-down constraints, re-compute their activations, and then return the set sorted by activation. Stymied by the second of these challenges, we turned away from trying to find ways of improving query-based retrieval with concurrency and started researching more radical alternatives for realizing massively concurrent retrieval processes. We quickly realized that two problems oppose the development of a memory system utilizing concurrent activation calculation:

P1. To parallelize activation calculation, one needs a language supporting concurrent computation. What language can do this for us?
P2. To continue allowing retrievals to be based on top-down retrieval constraints, we have to integrate the processing of top-down information with the process of concurrently computing chunk activations. How can a retrieval process utilizing concurrent activation calculation use top-down information and constraints?

Concurrently Computing Activations in Erlang The semantic anchoring of RML1 is currently realized in a modeling and simulation framework developed using the Erlang programming language (Armstrong, 2007; Cesarini & Thompson, 2009). Erlang is an open-source general-purpose functional programming language developed by Ericsson. Erlang is chiefly used to develop persistent, fault-tolerant, dynamically re-configurable, soft real-time constrained control systems that use large databases. Furthermore, it supports multiple process threads and automatically exploits multi-core and networked computing resources. In Erlang, program components are represented as sets of separate parallel threads. Erlang manages threads through a middleware framework called the Open Telecom Platform (OTP) which simplifies the development and execution of programs consisting of large numbers of concurrent processes. Programs written in Erlang can contain *millions* of concurrent processes (Armstrong, 2007).

RML1's Erlang-based semantic anchoring represents declarative knowledge in OWL-compatible ontologies (Smith, Welty, & McGuiness, 2008) that describe the classes, class properties, object properties, data properties, and instances constituting a domain. Each node in a semantic network is realized as a separate OTP process thread in Erlang. These process threads maintain information about: (a) retrieval parameters; (b) reference histories; (c) last activation level; (d) lists of class, object, and data relations constituting the defining properties of the individual; and (e) lists of object relations the individual serves a range role in. Process threads also receive and respond to messages sent to them by OTP supervisor processes. Each individual process thread is capable of responding to requests to re-compute and report their activation. Activity spreads in RML1 semantic networks as messages are asynchronously exchanged between the process threads constituting their nodes. Since process threads in Erlang execute concurrently, spreading activation achieved through asynchronous message passing and activation re-computing are massively parallel. The retrieval of declarative knowledge from a RML1 semantic network involves all concurrent multi-core computation available.

In order to maximize the parallelization of the activation computation, retrieval in the RML1 declarative memory system is based solely on the spread of activation in semantic networks. At first blush, it is not obvious how something functionally equivalent to an ACT-R top-down "isa" constraint can be obtained through bottom-up spreading activation. The following discussion explains how this is accomplished.

Replicating Top-Down Constraints with Message Filters and Endogenous/Exogenous Message Sources Table A1 (in Appendix) shows how the behavior of top-down retrieval request patterns in ACT-R can be replicated in RML1. Deliberate retrieval constraints introduce top-down network activity into semantic networks as endogenous messages. Endogenous messages introduce network activity into semantic networks but do not convey weighted activation to nodes and therefore do not influence a receiving node's calculation of its activation. Contexts introduce bottom-up network activity into semantic networks as exogenous messages. Exogenous messages function just like spreading activation in ACT-R; network activity introduced into semantic networks by exogenous sources convey weight and fan and therefore do influence a receiving node's calculation and reporting of its activation. Message filters prevent network activity from being sent to nodes lacking defining properties corresponding to the properties in them. For example, the "k1,v1" message filter in example 3 of Table A1, prevents the endogenous message "type,c1" from passing network activity into nodes lacking the "k1,v1" property.

Retrieval in RML1 proceeds in the following way:

1) An OTP supervisor process sends, in parallel, *"spread network activation"* endogenous and/or exogenous messages to nodes serving domain roles in the relations expressed in the messages that pass any present message filters. For example, in example 1 of Table A1, the OTP supervisor process will send a message to c1. Since "type,c1" is an endogenous message in this circumstance, the message will convey a weight of 0.

2) Nodes receiving "spread network activation" messages relay them, in parallel, to instances serving domain roles in relations with them. In example 1 of Table A1, any node serving a domain role in the "type,c1" relation will receive network activation. As mentioned earlier, individuals maintain lists of the relations they participate in with other individuals. Instances receiving these messages store the weighted activation increments they contain and notify the OTP supervisor that their activation has been influenced by network activity. Because "weights of activation spread" incorporated into endogenous supervisor messages are 0, stored activation increments from endogenous sources force the individual to re-compute their activation but do not increase spreading activation. If, as is the case in example 2 of Table A1, context produced an exogenous message "k2,v2", the "weight of activation spread" incorporated into exogenous supervisor messages would reflect attentional weight and fan.

3) The OTP supervisor process sends, in parallel, *"report your re-computed activation"* messages to nodes that reported contributions to their activations. Individual processes receiving these messages concurrently re-compute their activation. Individuals that received only messages containing 0 weights of activation spread report activation values based solely on changes to their base level activations.

4) Finally, the OTP supervisor posts the defining properties of the node reporting the highest activation to RML1's working memory.

Retrieval in a Large Declarative Memory

To determine the impact of concurrency in RML's retrieval process, a basic comparison study was conducted. In this comparison study, the wall-clock retrieval times of ACT-R and RML1 executing retrievals in large declarative memories were compared. To stress test the declarative systems of ACT-R and RML1, portions of the Moby Thesaurus II synonym database were transcribed into ACT-R's declarative memory and RML1's semantic network. The Moby Thesaurus II contains 30,260 root words that are related to each other by 2,520,264 synonyms. Compound root words were excluded from the comparison study. This exclusion process reduced the number of root words to 24,890. Five different declarative memory sets were created using this reduced set. Sets consisted of proportions of the reduced set of root words and the synonyms relating them. Table 2 below summarizes the properties of these sub-sets, and Figure 2 represents a portion of the smallest of these sub-sets.

Table 2: Properties of the synonym sets used in the comparison study.

Proportion	20%	25%	33%	50%	100%
Synonyms	53,560	77,313	145,073	318,435	1,281,763

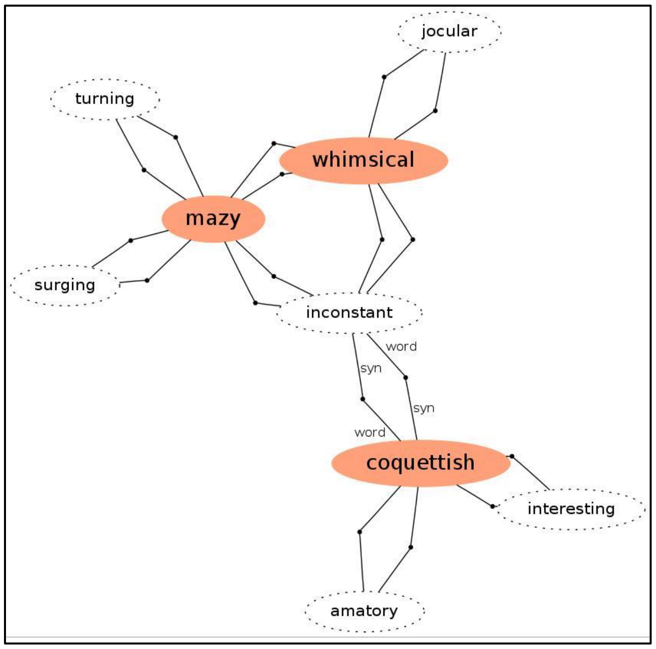

Figure 2: Portion of the Moby II semantic network showing a subset of the root words and synonyms related to the root words "coquettish", "mazy", and "whimsical". 29, 52, and 67 word/syn relations involving coquettish, mazy and whimsical are not shown.

To create a declarative memory in ACT-R, instances of a *root_word* chunk-type were used to represent root words and instances of a *synonym* chunk-type were used to represent word/synonym relationships between root words. Figure 3 shows chunk types and chunk instances that would allow an ACT-R model to represent and process some of the root words and relations displayed in Figure 2. To create an ontology-based semantic network in RML1, *root_word* and *synonym* classes were defined. Object properties necessary to relate words to *syn* in *synonym* instances were also defined. Figure 3 shows the definitions of the *root_word* and *synonym* classes and definitions of employed object and data properties. Representing these in an ontology allows RML1's runtime environment to search the semantic network and make inferences about arbitrary descriptions or entities lacking class identifiers.

```
(chunk-type root_word name)
(chunk-type synonym word syn)
(add-dm
  ...
  (coquettish ISA root_word name "coquettish")
  (inconstant ISA root_word name "inconstant")
  (flighty    ISA root_word name "flighty")
  (mazy       ISA root_word name "mazy")
  (whimsical  ISA root_word name "whimsical")
  ...
  (syn1 ISA synonym
        word coquettish
        syn flighty)
  (syn2 ISA synonym
        word coquettish
        syn inconstant)
  (syn3 ISA synonym
        word flighty
        syn mazy)
  ...
)
(set-all-base-levels 7 0)
```

```
{class, {root_word, [{subclass_of, thing}]}}.
{class, {synonym, [{subclass_of, relation}]}}.

{object_property,
  {word, [{sub_property_of, base_object_property},
          {domain, synonym}, {range, root_word}]}}.
{object_property,
  {syn, [{sub_property_of, base_object_property},
         {domain, synonym}, {range, root_word}]}}.
{data_property,
  {name, [{sub_property_of, base_data_property},
          {domain, root_word}, {range, string}]}}.

{individual,
  {coquettish, [{type, root_word}], [],
               [{name, "coquettish"}], 7}}.
{individual,
  {inconstant, [{type, root_word}], [],
               [{name, "inconstant"}], 7}}.
{individual,
  {mazy, [{type, root_word}], [],
         [{name, "mazy"}], 7}}.
{individual,
  {whimsical, [{type, root_word}], [],
              [{name, "whimsical"}], 7}}.

{individual,
  {s1, [{type, synonym}],
       [{word, coquettish}, {syn, inconstant}], [], 7}}.
{individual,
  {s2, [{type, synonym}],
       [{word, inconstant}, {syn, coquettish}], [], 7}}.
{individual,
  {s3, [{type, synonym}],
       [{word, mazy}, {syn, whimsical}], [], 7}}.
{individual,
  {s4, [{type, synonym}],
       [{word, whimsical}, {syn, mazy}], [], 7}}.
```

Figure 3: ACT-R (top) and RML1 (bottom) root_words and synonyms matching some of the Figure 2 information. Note the object and data property specifications in RML1 .

Equipment

A Dell Precision T7500 was used in the comparison study. The Dell's physical configuration included 2 quad core Intel 3.33Ghz Xeon (W5590) CPUs and 48 GiB of RAM. The computer's software configuration included the openSUSE 11.2 Linux-based OS, SBCL 1.0.35 running ACT-R6 r845, and Erlang R13B04.

Procedures

Context-sensitive retrievals of chunks from the sub-sets of the Moby Thesaurus II were carried out in ACT-R and RML1 using the request patterns and context representation shown in Table A2. Real-time costs of executing retrievals in ACT-R were measured by: (1) placing three chunks corresponding to root word chunks into slots of a goal chunk representing retrieval context; (2) initiating a retrieval request corresponding to the "+retrieval> isa synomym" request pattern; and (3) measuring elapsed system time until the retrieval process returned a chunk. The real-time costs of executing retrievals in RML1 were measured by: (1) distributing messages from endogenous and exogenous message sources that passed through message filters into the semantic network; and (2) measuring elapsed time until the OTP supervisor process managing the retrieval determined the network node with the highest activation.

Results

The same retrieval parameters were used in both systems: maximum associative strength was set to 5.0, the base-level constant was set to 0, and the base-level learning rate was set to 0.5. All chunks were initialized with 7 references.

Retrievals executed through ACT-R and RML1 returned the same synonym chunks, computed equivalent chunk activations, and retrieval latencies. The use of the "isa synonym" constraint in the ACT-R retrieval pattern required that the activations of all synonym chunks be calculated before the retrieval process could finish. Treating "type, synonym" as if it were from an endogenous message in the RML1 retrieval process correspondingly lead to all synonym instances re-computing and reporting their activations. Table 3 summarizes the results of the comparison study.

Table 3: ACT-R and RML1 performance. Times (seconds) represent average wall-clock time to execute 10 retrievals.

	20%	25%	33%	50%	100%
Synonyms	53,560	77,313	145,073	318,435	1,281,763
ACT-R	3.22	6.00	18.63	86.39	NA
RML1	0.44	0.64	1.21	2.65	10.90

The most important thing to notice in Table 3 is that while ACT-R (SBCL) performance time is increasing at a rate faster than the increase in chunks, RML1 (Erlang) is essentially scaling linearly. Added concurrency from additional processor cores will further improve the relative performance of RML1.

Conclusion

The declarative system underneath RML1 discussed in this paper is interesting because it: (1) does not depend on a top-down retrieval process that functions like a query against a relational database followed by activation calculation; (2) is capable of producing behavior that is functionally indistinguishable from ACT-R; (3) exploits concurrency in Erlang and therefore scales nearly linearly; (4) is part of the runtime environment supporting RML1, the first DSML researched and developed by the LSCM initiative. If cognitive modeling is to successfully grow in scope and complexity, it must find effective ways of meeting the challenges associated with maintaining and using large declarative memories. RML1's declarative system illustrates how concurrent knowledge activation calculation in large declarative memories can be technically realized and is therefore progress towards meeting LSCM challenges associated with modeling human memory on a large scale.

Acknowledgements

This research was funded through a seedling grant from the Chief Scientist of the 711th Human Performance Wing of the Air Force Research Laboratory awarded to Drs. Douglass & Myers, and through AFOSR grant #10RH05COR awarded to Dr. Douglass.

References

Anderson, J. R. (2007). How Can the Human Mind Occur in the Physical Universe? Oxford: OUP

Anderson, J. R., Bothell, D., Douglass, S. A., Byrne, M. D., Lebiere, C., Qin, Y., et al. (2004). An integrated theory of the mind. Psychological review, 111(4), 1036–1060.

Armstrong, J. (2007). Programming Erlang: Software for a Concurrent World. Raleigh: The Pragmatic Bookshelf.

Cesarini, F., & Thompson, S. (2009). Erlang Programming. O'Reilly Media, Inc.

Douglass, S. A., Ball, J., T., & Rogers, S. (2009). Large declarative memories in ACT-R. In A. Howes, D. Peebles, R. Cooper (Eds.), 9th International Conference on Cognitive Modeling – ICCM2009, Manchester, UK.

Smith, M. K., Welty, C., & McGuiness, D. L. (2008). OWL Web Ontology Language Guide. W3C.

Appendix

Table A1. Examples of how query-based retrieval behavior in ACT-R can be replicated using message passing in RML1 semantic networks. The character "*" is used in messages to represent a wildcard that is free to match against any relation. The "*" is necessary because contextual priming in ACT-R is insensitive to the key component of the key/value pairs in context chunks. Notice that examples 3 and 4 yield the same retrieval behavior while using the "type,c1" and "k1,v1" messages in different ways. Since it is likely to be the case that the fan of v1 is less than the fan of c1, treating the "k1,v1" message as endogenous will greatly reduce the spread of network activity and therefore expedite retrieval.

Example	ACT-R		RML1		
				Message Sources	
	Retrieval Request	Context	Message Filters	*Exogenous*	*Endogenous*
1	+retrieval> isa c1				type,c1
2	+retrieval> isa c1	isa c2 k2 v2		k2|*,v2	type,c1
3	+retrieval> isa c1 k1 v1		k1,v1		type,c1
4	+retrieval> isa c1 k1 v1		type,c1		k1,v1
5	+retrieval> isa c1 k1 v1	isa c2 k2 v2	type,c1	k2|*,v2	k1,v1

Table A2. ACT-R retrieval requests and contexts & RML1 message filters and message sources employed in the declarative memory system comparison study. To ensure the fairness of the comparison, all exogenous messages conveying activation due to contextual priming had to be insensitive to relation (they all had to use "*"). Parenthesized numbers indicate fan.

Example	ACT-R		RML1		
				Message Sources	
	Retrieval Request	Context	Message Filters	*Exogenous*	*Endogenous*
1	+retrieval> ISA synonym	=goal> c1 whimsical(73) c2 mazy (60) c3 coquettis(31)	type,synonym	*,whimsical *,mazy *,coquettish	type,synonym
2	+retrieval> ISA synonym	=goal> c1 vexing (20) c2 heavy (249) c3 operose (42)	type,synonym	*,vexing *,heavy *,operose	type,synonym
3	+retrieval> ISA synonym	=goal> c1 entangle (63) c2 stare (65) c3 woo (33)	type,synonym	*,entangle *,stare *,woo	type,synonym