Modified Sparse Distributed Memory as Transient Episodic Memory for Cognitive Software Agents*

Uma Ramamurthy, Sidney K. D'Mello and **Stan Franklin**
Computer Science Division and Institute for Intelligent Systems,
The University of Memphis, Memphis, TN 38152, USA.
urmmrthy@memphis.edu, sdmello@memphis.edu, and franklin@memphis.edu

Abstract - *This paper presents a Modified Sparse Distributed Memory architecture for use in software agents with natural language processing capabilities. We have modified Kanerva's Sparse Distributed Memory (SDM) into an architecture with a ternary memory space. This enables the memory to be used in IDA, the Intelligent Distribution Agent built for the U.S. Navy. IDA implements Baars' global workspace theory, a psychological theory of consciousness. As a result, it can react to novel and problematic situations in a more flexible, more human-like way than traditional AI systems. IDA performs a function, namely billet assignment, heretofore reserved for humans. We argue that such flexibility requires advanced memory systems such as transient episodic memory and auto-biographical memory. Here, we present the architecture, tests and results of this modified SDM system which can be used as a transient episodic memory in suitable software agents.*

Keywords: Transient Episodic Memory, Cognitive Modeling, Software Agents, Modified Sparse Distributed Memory.

1 Introduction

An autonomous agent is a system situated in, and part of an environment. It senses that environment, and acts on it, over time, in pursuit of its own agenda [11]. Cognitive Agents are autonomous agents that have cognitive features like multiple senses, perception, working memory, transient episodic memory, declarative memory, attention, planning, reasoning, problem solving, learning, emotions, moods, attitudes, multiple drives, etc., [8]. One way to design such cognitive agents is to model them after humans. We have designed and implemented such cognitive agents within the constraints of the Global Workspace theory of consciousness, a psychological theory that gives a high-level, abstract account of human consciousness and broadly sketches its architecture [2], [3]. We call such agents "conscious" software agents.

Global workspace theory postulates that human cognition is implemented by a multitude of relatively small, special purpose processors, almost always unconscious. Coalitions of such processors find their way into a global workspace and hence into consciousness. This limited capacity workspace serves to broadcast the message of the coalition to all the unconscious processors, in order to recruit other processors to join in handling the current novel situation, or in solving the current problem. Conscious software agents should implement the major parts of the theory, and should always stay within its constraints. IDA is one such "conscious" software agent.

1.1 IDA's Architecture

IDA (Intelligent Distribution Agent) is a cognitive software agent [8] developed for the U.S. Navy. At the end of each sailor's tour of duty, he or she is assigned to a new billet by a person called a *detailer*. IDA's task is to facilitate this process, by completely automating the role of a detailer. The IDA model [9] has a number of different memory systems, including working memory, transient episodic memory and auto-biographical/declarative memory. Some of these memories are motivated by Sparse Distributed Memory [12]. The focus of this paper is on a transient episodic memory for software agents such as IDA.

2 Transient Episodic Memory

Episodic memory is for events having features of a particular time and place [5]. This memory system is associative in nature and content-addressable. It has been proposed that working memory probably includes an episodic buffer that can hold episodic information for a short duration [4].

Humans have a content-addressable, associative, transient episodic memory with a decay rate measured in hours [6], [10]. Humans are able to recall in great detail events of the current day – where they park their cars, whom they met that morning, what they discussed, what they had for meals, etc. These details of the events/episodes stay with us only for short durations – for some hours. We hypothesize that for cognitive agents to recall such details of episodes while they interact with and adapt to their dynamic environments, they need a transient episodic memory (TEM). The IDA cognitive model has a TEM with a decay rate measured in hours. We hypothesize that information stored in this memory

* 0-7803-8566-7/04/$20.00 © 2004 IEEE.

which has not decayed away, is consolidated into declarative memory at certain intervals. In this paper, we describe a modified SDM architecture that works as a TEM for cognitive software agents such as IDA. We present the implementation and test results of such a modified SDM architecture.

2.1 Sparse Distributed Memory

Transient episodic and declarative memories have distributed representations in IDA. There is evidence that this is also the case in the animal nervous systems. In IDA, these two memory systems are implemented computationally using a modified version of Kanerva's Sparse Distributed Memory (SDM) architecture [12], [1].

SDM implements a content addressable random access memory. Its address space is enormous, of the order of 2^{1000}. Of this space, you choose a manageable, uniform random sample, say 2^{20}, of allowable locations. These are called hard locations. Thus the hard locations are sparse in this address space. Many hard locations participate in storing and retrieving of any datum, resulting in the distributed nature of this architecture. Hamming distance is used to measure the distance between any two points in this memory space.

Each hard location is a bit vector of length 1000, storing data in 1000 counters with a range of -40 to 40. Each datum to be written to SDM is a bit vector of length 1000. Writing 1 to a counter results in incrementing the counter, while writing a 0 decrements the counter. To write in this memory architecture, you select an access sphere centered at location X. So, to write a datum to X, you simply write to all the hard locations (roughly a 1000 of them) within X's access sphere. This results in distributed storage. This also provides naturally for memory rehearsal – a memory trace being rehearsed can be written many times and each time to about 1000 locations.

To read/retrieve from location Y, you compute the bit vector read at Y – by assigning its k^{th} bit the value 1 or 0, based on Y's k^{th} counter being positive or negative. Hence each bit of the bit vector read at Y is a majority rule decision of all the data that have been written at Y. Effectively, the read data at Y is an aggregate of all data that have been written to Y, but may not be any of them exactly. Similar to writing, retrieving from SDM involves the same concept of access sphere – you read all the hard locations within the access sphere of location Y, pool the bit vectors read from all these hard locations and let each of the k^{th} bits of those locations participate in a majority vote for the k^{th} bit of Y. Effectively, you reconstruct the memory trace in every retrieval operation.

This memory can be cued with noisy versions of the original memory trace. To accomplish this, you employ iterated reading – first read at Y to obtain the bit vector, Y1. Next read at Y1 to obtain the bit vector Y2. Next read at Y2 to obtain the bit vector, Y3. This is shown in Figure 1 below. If this sequence of reads converges to Y', then Y' is the result of iterated reading at Y. Convergence happens very rapidly in this architecture, while divergence is indicated by an iterated read that bounces erratically and out of the access sphere in this address space.

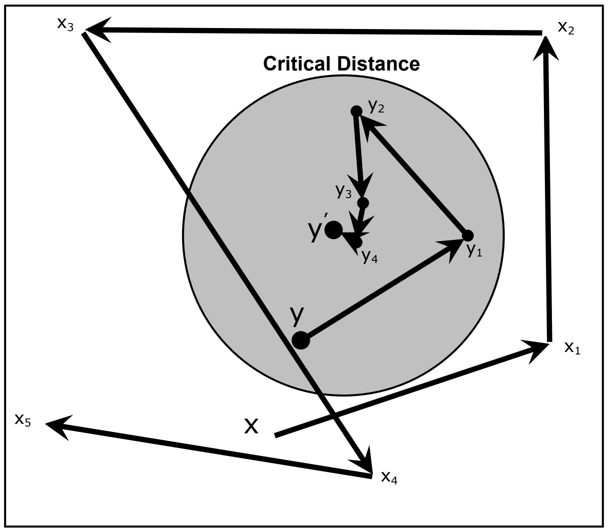

Figure 1. Converging/Diverging Reads [12]

Memory recall is a constructive process, supported by evidence from psychology and imaging studies [14]. The SDM architecture has several similarities with human memory [12] and provides for "reconstructed memory" in its retrieval process.

2.2 Rationale for the Modified SDM

Experiments on our implementation of Kanerva's original SDM for cognitive agents such as IDA in a text-based domain indicated the need for an architecture modification. When events are unfolding, the feature vector is not always complete. So, more often, the agent has to write partial feature-sets to its memories. Similarly, when the agent cues its memory for retrieval, the read-cues are often partial feature-sets.

We propose a modification to a ternary memory space while maintaining a binary address space for the hard locations. Adding "don't cares" (*) to the 0's and 1's of binary space yields a ternary memory space. This will accommodate flexible cuing with fewer features than the actual memory trace where missing features are represented by "don't cares" (*). An adjustment was made to Hamming distance calculations such that the distance between a "don't care" (*) and a 0 or 1 was set to (0.5).

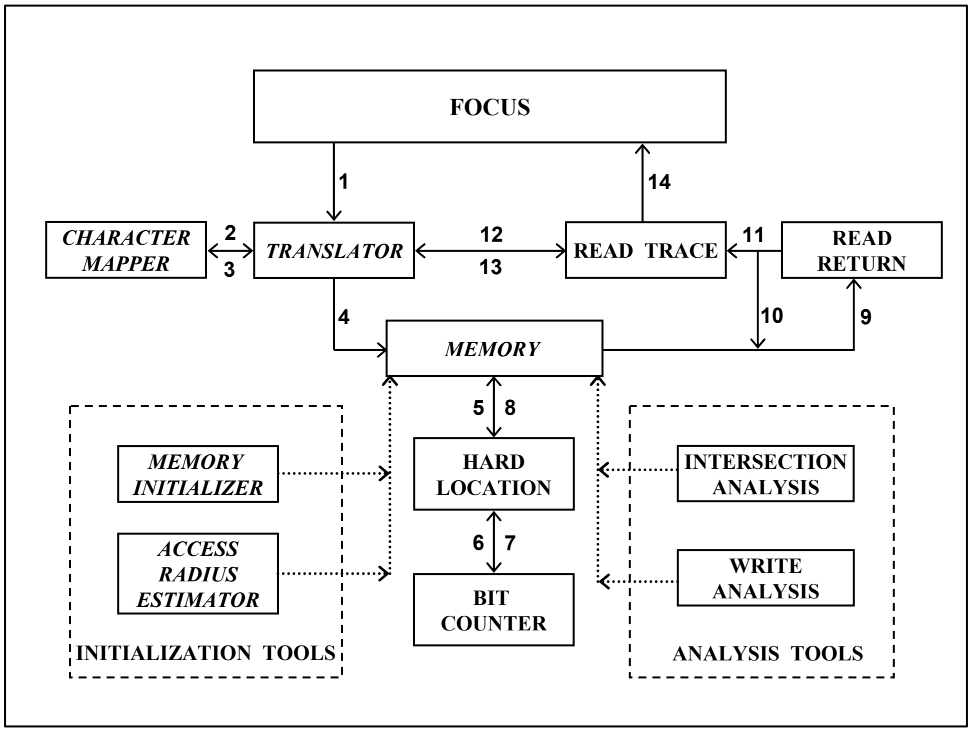

Figure 2. The Modified SDM Architecture (Abstract classes are italicized, steps are numbered)

This modification to the memory space also addresses two essential features of episodic memory systems [14]. Episodic memory systems must have binding-error detectors and binding-error integrators. Given an event, the episodic memory trace must respond not only to partial cues, it should also be capable of separating a memorized event from very similar events.

Recalling a similar event, but with an explicit feel for the mismatch is a salient feature of episodic memory. For example, when the agent is cued with "Anthony invited his friend *Barbara* to brunch in the canteen at 12-noon", it retrieves a similar memorized event: "Anthony invited his friend *Brianna* to lunch in the cafeteria". In the IDA model, when a cue retrieves a similar event but not the original memorized event, the *attention mechanism* watching the retrieved trace brings it to "consciousness" and thus, the agent is aware of the mismatch [9].

With the original SDM, testing for binding-error detectors and integrators in the IDA model's textual domain produced minimal results. As will be seen in the Experimental Analysis section, the Modified SDM produces an order of magnitude improvement in performance. For software agents that have sensory inputs in natural language, we argue that such a Modified SDM provides a flexible and robust way to represent domain knowledge of the agent for storage and retrieval. In our design and implementation of this system, we have ensured a generic design that can easily be tailored for use in other cognitive software agents irrespective of the domains they 'live' in.

3 Design and Implementation

The Modified SDM software was designed to be general enough to facilitate its use as a transient episodic memory in a variety of software agents. The design follows the Object Oriented Paradigm and several of the classes are abstract to enable future architectural extensions. A simplified version of the design is presented in Figure 2 above.

3.1 Core Modules

The Focus is the central point of interaction between the memory and the outside world. It is an interface whose implementation determines the behavior of the memory. Therefore, integrating the Modified SDM software into new domains is as simple as implementing a new Focus.

A Bit Counter counts the frequencies of 1's, 0's and *'s ("don't cares") for a single dimension for a single hard location. A vector of ternary Bit Counters and a binary address space constitute a Hard Location. The Memory itself is a set of Hard Locations.

A Character Mapper maps a single character into a set of bits (binary or ternary). A Translator converts the memory-write or read-cue into a content addressable vector as specified by its internal Character Mapper.

A Read Return maintains pertinent information for a single read iteration. The Read Trace stores a predefined number of Read Returns (usually 10 or fewer in case of convergence or divergence) and can recreate the entire trace for offline analyses.

3.2 Initialization and Analysis Modules

The Memory Initializer is an abstract class which can be extended to allow different initialization techniques such as a random, uniform, or domain based distribution of the hard locations. The Access Radius Estimator is used to tune the access radius for an initialized memory.

The Intersection Analysis tools implement an algorithm that efficiently measures the intersection between the access spheres for different memory-writes. The Write Analysis tool periodically monitors the memory and records various statistics regarding the activity of the hard locations.

3.3 Writing and Reading

For writing, the Focus intercepts the memory write and sends it to the Translator who creates a bit vector by consulting its Character Mapper. The Bit Counters of the Hard Locations within the access sphere of the vector are updated. The writing process can be traced by Steps 1 – 6 in Figure 2.

The first 5 steps for reading are identical to writing. In steps 6-8, a Read Return object is created from the retrieved vector obtained by pooling the Hard Locations within the access sphere for the bit vector of the read-cue. Convergence or divergence testing is then performed and the vector may be sent back to the Memory for the next iteration (Step 10). It is also registered to the Read Trace (Step 11) and is converted back into text (Steps 12 – 13).

4 Experimental Analysis and Results

For testing this Modified SDM, we chose a case-grammar based feature vector [7]. The focus of our testing was to evaluate the Modified SDM's performance when the memory writes are done with either complete or partial feature sets. Retrieval was tested with fully specified read-cues, partial read-cues and finally, read-cues with a feature replaced with an incorrect feature in order to test binding-error detection, over both types of writes.

4.1 Experimental Setup

The memory was randomly initialized with 10,000 hard locations as earlier testing with a varying numbers of hard locations ranging from 500 to 50,000 did not show a significant performance improvement after 10,000. The dimensionality of the memory space was set at 448 based on the case-grammar template used for the testing. The case-grammar template selected is illustrated in Figure 3, with examples of fully specified feature-sets and partially specified feature-sets representing episodes.

4.2 Tests Conducted

Two issues related to transient episodic memory were addressed in the testing phase, retrieval with partial cues, and retrieval with cues in which a feature is replaced with an incorrect feature to simulate a binding-error.

TEMPLATE:
"Agent | Verb | Recipient-Adjective | Recipient | Object-Adjective | Object | Place | Time"

EXAMPLES:
TESTS A & A′: "Richard | drives | joyful | Vanessa| lively | comedy | Theatre | Friday"

TESTS B & B′: "Richard | drives | * | * | lively | comedy | * | Friday"

Figure 3. Case-grammar template & example episodes of fully-specified and partial feature sets

The tests were divided on the basis of two types of writes made to memory. The first set (Test-A in the original SDM and Test-A′ in the modified SDM) consisted of completely specified feature-sets. An example episode: "Nathan accepts elated Michael venture scheme eatery Tuesday". The second set (Test-B in the original SDM and Test-B′ in the modified SDM) of writes was composed of partial feature-sets with 75 percent of the features specified. "Nathan accepts * Michael venture scheme eatery *" is an example of a partial memory-write for the second set. Here, the *recipient-adjective* and the *time* features have been replaced with "don't cares" (*).

Five different types of read-cues with varying percentages of missing features were constructed to evaluate the retrieval from memory for each of these two sets of writes. The first set (R1) consists of complete read-cues that were identical to the writes. The second, third and fourth sets (R2, R3 and R4) were partial read-cues that consist of 87.5%, 75% and 62.5%-50% of the feature-sets respectively. An example of one of these partial read-cues is "Nathan accepts * Michael venture * eatery *" which contains 62.5% of the feature-set.

The above four sets of read-cues addressed the first issue of retrieval with partial cues. The fifth set (R5) of read-cues had a feature replaced with an incorrect feature to simulate a binding-error. For example, "Andrew accepts elated Michael venture scheme eatery Tuesday" would be the read-cue for the fully specified write mentioned above. Here the *agent* feature 'Nathan' is replaced with an incorrect feature 'Andrew'. Each of the five sets of read-cues was tested for retrieval on both sets of writes for both the original and the Modified SDM.

4.3 Results and Discussion

The tests were evaluated by the number of features fully recovered in the retrieval. A feature was considered to be fully retrievable if no more than two of its characters were incorrect and hence can be recovered fully with post-processing. The scoring of the retrieved episodes was based on the following scale: (1) all features fully retrieved scored as 1.0; (2) all but one feature fully retrieved scored as 0.75; (3) all but two features fully retrieved scored as 0.5; (4) retrievals with more than two irretrievable features or diverged reads were scored as 0. The overall score for each test was the average over the total number of episodes written.

Table 1 shows the scores for five sets of read-cues -- R1 for complete read-cues, R2-R4 for partial read-cues and R5 for binding-error detection. The plot in Figure 4 shows the performance comparison between the original SDM and the Modified SDM for the four sets of read-cues, namely R1-R4.

Table 1. Test Scores

Read-Cue Sets	Original SDM		Modified SDM	
	Test-A	Test-B	Test-A′	Test-B′
R1	0.958333	0.986111	0.972222	1
R2	0.569444	0.097222	0.75	0.972222
R3	0.069444	0	0.513889	0.916667
R4	0	0.055556	0.333333	0.666667
R5	0.263889	0.069444	0.555556	0.930556

Extension of the content-space to include the "don't cares" (*) provides a significant improvement as the percentage of missing features in the read-cues increases to a reasonable degree. For Test-A′, the performance score dropped from 97% to 33% as the percentage of missing features in the read-cues increased from zero to 50 percent. The performance score dropped from 100% to 67% for Test-B′ as the percentage of missing features in the read-cues increased from zero to 50 percent. This indicates that doing partial-writes is advantageous as "don't cares" effect more rapid convergences, due to the modification to the Hamming distance calculation.

This is also true in the case of binding-error detection as can be seen from the scores (R5). In Test-B′, the performance score was much higher (93%) compared to Test-A′ (56%).

Table 1 also shows that for fully specified feature sets in both the writes and read-cues (R1), there is very little performance difference between the original SDM and the Modified SDM. Although Table 1 indicates that Test B performed slightly better for R4 than for R3, this is attributed to random initialization. The Modified SDM exhibits better performance in comparison to the original SDM with partial writes and partial read-cues.

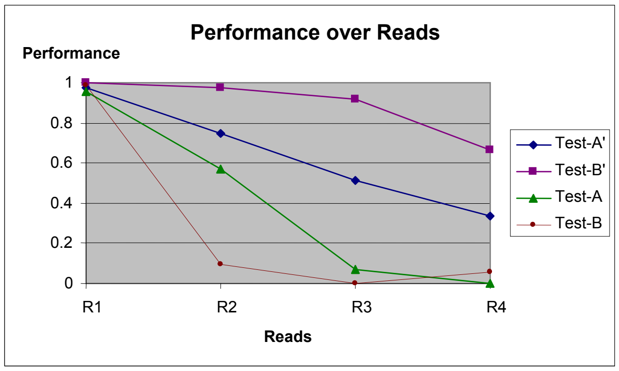

Figure 4. Performance of Modified SDM vs. Original SDM

The Modified SDM showed clear convergences (illustrated in Figure 1) both with partial read-cues and with binding-errors in read-cues. We saw interesting associations between related episodes. Figure 5 shows an

```
Read-Cue= * informs * Nathan * * * Tuesday
Target = Nathan accepts elated Michael venture scheme eatery *

41.5   * kowwms * Niwhand * swmwmmp * uwemo
32.5   * oowee` * Nighcne * swuwep giwurmp uwemo
44.5   Fithcna eokoee  * Nachan` * sgufew  Fiwurm` uwemo
41.0   Fathan` aocnee` kowwm~ Nachag` * sofgee gaterm uwemo
21.5   Nathan` ancgqe` cmuumd  Nachag` * sabgee eatery uwemo
19.0   Navhan` ancgqes cmuumd  Nichag` w sa`gee eatery uwemo
18.5   Nathan` aocgpes cmqumd  Oichagh wew}we sa`gee  eatery
20.5   Nathan  aocepes amqumd  Michael veovure sc`eee  eatery
18.0   Nathan  accepus emated  Michael venture scheme eatery  *
 2.5   Nathan  accepts elated  Michael venture scheme  eatery  *
```

Figure 5. Convergence with a highly partial read-cue

example of one such iterated read trace of a highly partial read-cue retrieving an associated episode. "Don't cares" are represented with "*" in the example run. The column of numbers is the modified Hamming distance between the iterated retrievals. The set of strings next to this column shows the iterative reads that resulted in the convergence.

Figure 6 shows an example of binding-error detection. The *agent* feature in the read-cue "Roberto" is an incorrect feature for *agent* "Richard" of the original memory trace, simulating the binding-error.

```
Read-Cue = Roberto drives joyful * lively * Theater Friday
Target   = Richard drives  joyful  * lively  * Theater Friday

24.5     Richard dbives  boyfel  * nmvemy  * Theater Drifai`
 2.0     Richard dbives  joyfel  * nmvem}` * Theater Drifai`
 1.0     Richard dbives  boyfel  * nmvgm}` * Theater Drifai`
 2.0     Richard dbives  boyfel  * lmvem}` * Theater Drifai`
 3.5     Richard dbives  joyfel  * nmvemy  * Theater Drifai`
 3.5     Richard dbives  joyfel  * livemy  * Theater Drifay`
 2.0     Richard dbives  joyfel  * livemy  * Theater Friday
 3.0     Richard drives  joyful  * livemy  * Theater Friday
 0.5     Richard drives  joyful  * lively  * Theater Friday
 0.0     Richard drives  joyful  * lively  * Theater Friday
```

Figure 6. Convergence with binding-error in read-cue

5 Future Work and Conclusions

In the current version of the Modified SDM, we have used random initialization as per Kanerva's original architecture [12]. Given the memory and computational constraints of the computing systems that software agents 'reside' in and hence, the constraints on the number of hard locations that such agents' TEM can have, we hypothesize that domain-based initialization mechanisms for the Modified SDM will further improve the system performance. Such focused initialization may well result in faster and better convergence in the retrieval process. In our future work, we will explore these mechanisms. Further, we plan to implement decay in TEM and implement the consolidation mechanism between the TEM and the declarative memory (based on the SDM architecture) of systems like IDA.

We have presented a Modified version of the Sparse Distributed Memory system for use in text-based software agents that require the cognitive functionality of transient episodic memory. This modified version promises significant improvement over the original SDM architecture for software agents that interact with their environments in natural language via email and other messaging systems.

6 Acknowledgement

The first author is supported in part by NIH Cancer Center Support CORE grant, P30 CA-21765 and by the American Lebanese Syrian Associated Charities (ALSAC). The authors acknowledge the support of Igor Beliaev, Dr. Junmei Zhu and the Conscious Software Research Group.

References

[1] Anwar, A., and S. Franklin. (2003) Sparse Distributed Memory for "Conscious" Software Agents. *Cognitive Systems Research* 4:339-354.

[2] Baars, Bernard J. (1988) "A Cognitive Theory of Consciousness," Cambridge: Cambridge University Press.

[3] Baars, Bernard J. (1997) "In the Theater of Consciousness," Oxford: Oxford University Press.

[4] Baddeley, A. D. (2000) The episodic buffer: a new component of working memory? *Trends in Cognitive Science* 4:417-423.

[5] Baddeley, A., M. Conway, and J. Aggleton. (2001) *Episodic Memory*. Oxford: Oxford University Press.

[6] Conway, M. A. (2001) Sensory-perceptual episodic memory and its context: autobiographical memory. In *Episodic Memory*, ed. A. Baddeley, M. Conway, and J. Aggleton. Oxford: Oxford University Press.

[7] Fillmore, C. (1968) The case for case. In *Universals in Linguistic Theory*, ed. E. Bach, and R. T. Harms. New York: Holt, Rinehart and Wilson.

[8] Franklin, Stan (1997) "Autonomous Agents as Embodied AI," Cybernetics and Systems' Special issue on Epistemological Aspects of Embodied AI, 28:6 499-520.

[9] Franklin, Stan (2001) "Conscious Software: A Computational View of Mind" in Soft Computing Agents: New Trends for Designing Autonomous Systems, ed. V. Loia, and S. Sessa. Berlin: Springer (Physica: Verlag).

[10] Franklin, S., B.J.Baars, U. Ramamurthy and M. Ventura. (In review) The Role of Consciousness in Memory.

[11] Franklin, Stan and Graesser, Art (1997) "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents," Proceedings of the Agent Theories, Architectures, and Languages Workshop, Berlin: Springer Verlag, 193-206.

[12] Kanerva, P. (1988) "Sparse Distributed Memory," Cambridge MA: The MIT Press.

[13] Ramamurthy Uma, Sidney K. D'Mello and Stan Franklin (2003) "Modeling Memory Systems with Global Workspace Theory", Seventh Conference of the Association for the Scientific Study of Consciousness - ASSC7, May 2003.

[14] Shastri, Lokendra (2002) *Episodic memory and cortico-hippocampal interactions*, in TRENDS in Cognitive Sciences, Vol. 6, No. 4, April 2002: 162-168.