EPISODIC MEMORY ASPECTS
LIDA ARCHITECTURE – WORLDSERVER 3D SCENERY

José Renato Borelli

Faculdade de Engenharia Elétrica e Computação
Universidade Estadual de Campinas
Campinas – São Paulo - Brasil

1 ABSTRACT.

This review is part of the final project for IA006 discipline at UNICAMP/FEEC/DCA, Prof. Ricardo Gudwin, http://faculty.dca.fee.unicamp.br/gudwin/courses/IA006 and shows the relevance of memory and learning capabilities as basis for cognitive architectures design and more refined features. The project implements a basic episodic memory on LIDA/WS3d scenery - therefore, some high-level episodic memory concepts and mechanisms are reviewed. The presented concepts focus on the studied architectures: SOAR, CLARION and LIDA.

Different disciplines such as psychology and neuroscience have been examining episodic memory, also referred to as declarative memory, for more than three decades. Now, engineering and computer science are developing an increasing interest in episodic memory for artificial systems. For robots and modern machines, it seems to be essential to collect autobiographical memories to improve action planning based on past experiences.

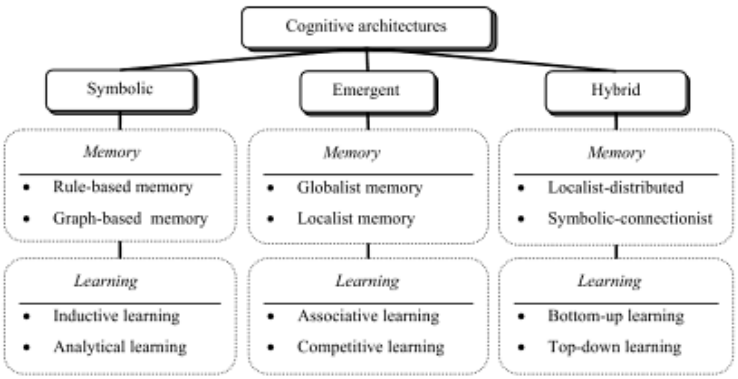
2 INTRODUCTION.

Cognitive architectures have been developed in order to model human performance in several situations of modern life. Adaptive behavior, dynamic behavior, flexible behavior, development, evolution, learning, knowledge integration, vast knowledge base, natural language, real-time performance and brain realization are some criteria for cognitive systems evaluation, proposed by Allen Newell, 1990 [1]. After that, some other evaluation criteria were designed, considering new concepts, refinements and architectures.

Two important key design properties that represent cognitive architectures development are memory and learning. Several types of memories are designed and optimized to their main task: repository for background knowledge about the world and oneself, episodic memory of events and activities, working memory to program actions, learning and organize knowledge.

Memory and Learning are capabilities that form the basic layer for cognition aspects of any architecture, above which more specialized functions and intelligent capabilities are built, such as reasoning, planning, flexibility, self-regulation, among others. Organization of memory is basically derived from knowledge representation schemes.

Cognitive architectures may be classified into models, based on their learning and memory design: *symbolic, emergent and hybrid.*



Symbolic architectures focus on information processing using high-level symbols or declarative knowledge, in a top-down analytic approach. Generally, a physical symbol system has the ability to input, output, store and alter symbolic entities, and act in order to reach its goals. The information flows from sensory inputs through working memory that access semantic memory by its executive functions for knowledge retrieval. Graph-based representations are typically encoded as directed graph structure comprising nodes for symbolic entities, their attributes and edges for relationships among them. **SOAR** (*State, Operator And Result*) is a classical example of expert rule-based cognitive architecture – it stores knowledge in form of production rules, arranged in terms of operations that act in the problem space.

Emergent architectures are inspired by connectionist ideas [3] - low-level activation signals flowing through a network consisting of numerous processing units, a bottom-up process relaying on the emergent self-organizing and associative properties. Processing elements form network nodes that interact with each other in a specific way changing their internal states and revealing interesting emergent properties. The Multi-Layer Perception (MLP) and other neural networks based on delocalized transfer functions process in a distributed and global way.

Hybrid architectures result from combining the symbolic and emergent paradigms. Symbolic architectures are able to process information and realize high-level cognitive functions, such as planning and deliberative reasoning. However, the major issues in this approach are the formulation of symbolic entities from low-level information, as well as the handling of a large amount of information and uncertainty. Emergent architectures are better suited for capturing the context and handling many pieces of low-level information simultaneously. Therefore, each architecture address the limitations of the other, allowing creation of a complete new architecture that covers all levels of processing – from stimuli to higher-level cognition. **CLARION** (*The Connectionist Learning Adaptive Rule Induction ON-Line*) is a hybrid architecture that incorporates a distinction between explicit (symbolic) and implicit (sub-symbolic) processes and captures the interactions between them. The design objective is to develop artificial agents for certain cognitive task domains and to understand human learning and reasoning processes in similar domains. CLARION architecture contains four memory modules, each comprising a dual explicit-implicit representation: action-centered subsystem (ACS), non-action-centered subsystem (NCS), motivational subsystem (MS) and metacognitive subsystem (MCS). Essentially, the ACS module serves to regulate the agent's actions, while NCS maintain the general system knowledge (explicit and implicity). MS provides a motivation/impetus for perception, action and cognition, while MCS monitors, directs and alters operations of the other three modules. CLARION also employs different learning methods for each level of knowledge. Learning of implicit knowledge is achieved using reinforcement learning methods such as Q-learning or supervised methods such as standard back-propagation, both of which can be implemented using an MLP network. The implicit knowledge acquired at the bottom level is used to elaborate the explicit knowledge at the top level via bottom-up learning. Top-down learning may also be achieved by rules at the top level and allowing the bottom-level to accumulate knowledge by "observing" actions guided by these rules – the system gradually becomes more dependent on the bottom level. **LIDA** (*The Learning Intelligent Distribution Agent*) is a conceptual and computational framework for intelligent, autonomous, "conscious" software agent that implements ideas of Global Workspace theory [4]. The architecture employs a partly symbolic and partly connectionist memory organization, with all symbols grounded in the physical world, interacted by distinct modules such as for perception, working memory, emotions, semantic memory, episodic memory, action selection, expectation and automatization (learning procedural tasks from experience), constraint satisfaction, deliberation, negotiation, problem solving, metacognition and conscious-like behavior. Most operations are done by codlets implementing the unconscious processors, specialized networks, of the Global Workspace theory. Episodic learning involves memorize specific events (what, where and when) resulted from events taken from the content of "consciousness" being encoded in the transient Episodic Memory. Procedural learning concerns learning of new actions and actions sequences to accomplish new tasks. There is no doubt that this architecture may explain and simulate many features of mind but it will always need more development and refinement.

As already mentioned, learning and memory capabilities are the bases for all architectures. This work intends to do a review about Episodic Memories structures and apply some of the concepts to a particular case – LIDA implementation.

2.1 LEARNING:

This report is about Episodic Memory and its implementation for a particular application using LIDA platform. Therefore, aspects regarding learning process will be introduced in another opportunity

2.2 EPISODIC MEMORY:

Engelkamp [5] propose to distinguish memory systems based on the type of stored information (e.g. episodic-semantic, verbal-nonverbal-imaginal), the type of processes involved (e.g. declarative-procedural, implicit-explicit) and such memory systems based on the length of time that information is retained (e.g. short–term-long–term). The study of episodic memory began in the early 1970s when the psychologist Endel Tulving made a first distinction between episodic and semantic memory [6]. At that time episodic memory (EM) was defined in terms of materials and tasks. Tulving specified episodic memory as your experiences of certain, spatio-temporal definite episodes (e.g. your last business trip) and our general knowledge (language translations, facts like "what is a pen") as the semantic memory (SM). However, his suggestion that episodic and semantic memory are two functionally different memory systems quickly became controversial. As a result of the criticism, the episodic memory definition was refined and elaborated in terms of its main ideas such as self, subjectively sensed time, and ***autonoetic consciousness***. Today, episodic memory is seen as one of the major neurocognitive memory systems [7] that is defined in terms of its special functions (what the system does or produces) and its properties (how it does that). It shares many features with semantic memory, which it grew out of, but it also possesses features that semantic memory does not have [8]. Episodic memory is oriented towards the past in a way in which no other kind of memory system is. It is the only memory system that allows people to consciously re-experience their past. It has a special and unique relationship with time [9].

The brain uses vast amounts of memory to create a model of the world. Everything a person knows and has learned is stored in this model. The brain uses this memory-based model to make continuous predictions of future events[10]. If those predictions are disproved, the brain learns (e.g. by novelty detection [11]), and adjusts its memories according to the new data. The memory seems to be organized in a hierarchy, each level being responsible for learning a small part of the overall model. Kanerva [12] proposed a sparse distributed memory (SDM) model that offers many of

the characteristics that a human memory possesses. He also developed a mathematical model for this theory.

Reviewing characteristics of episodic memory in humans it can be listed some inspiring:

Autonoetic: Remembering episodic memory is characterised by a state of awareness unlike that in semantic memory that is noetic. When one recollects an event autonoetically, one re-experiences aspects of a past experience. Re-experiencing of an already learnt episode is not necessary.

Autobiographical: A person remembers an episode from his or her own perspective. There is no possibility to change the viewpoint in AI systems. To put oneself in someone else's place is the highest achievement of human intelligence. Moreover, there are studies proving that autobiographical and episodic memory are separate memory systems [13].

Variable Duration: The time period that is spanned by an episode is not fixed.

Temporally Indexed: The "rememberer" has a sense of the time at which the remembered episode occurred.

Imperfect: Our memory is incomplete and can have errors. New sensations are forced to satisfy already experienced concepts.

Primed: Recall occurs more quickly when it is primed by repetition, recall of related information, or similar states.

Forgetting: It is still not clear if forgetting is a problem of actual information loss in long-term memory (LTM), or rather a problem of recall of the memory traces. Currently, mechanisms of active forgetting are being discussed [14].

Level of Activation: Exposure frequency and recency affect the speed and probability of recall. The level of activation mainly describes the primacy & recency effect where the former is based on LTM effects and the latter is based on the contents of the working memory.

Computationally, mechanisms of episodic memory can be used to develop new learning algorithms and experience-based prediction systems. Agents that do not remember their past are bound to repeat both the previous mistakes and the reasoning efforts behind them. Thus, using an episodic memory helps to save time by remembering solutions to previously encountered problems and by anticipating undesirable states. In literature several important approaches to creating episodic memory in artificial systems have been explored. Computational models of episodic memory can be divided into two categories: abstract and biological.

Abstract Models:

Make claims about the "mental algorithms" that support recall and recognition judgments, without addressing how these algorithms might be implemented in the brain.

SOAR-EM - Nuxoll & Laird extend the CBR paradigm by integrating episodic memory with a general cognitive architecture and developing task independent mechanisms for encoding, storing, and retrieving episodes [16]. They extend SOAR, one of the major cognitive architectures based on production rules [17]. SOAR has two types of knowledge, working memory (short-term, declarative) and production rules (long-term, procedural) and has been extended with episodic memory mechanisms into SOAR-EM. In previous articles they propose a Packman-like domain to wander around in a limited grid and collect the most food-points in the least amount of time. Their goal was for the agent to use its episodic memory in place of its knowledge about the food-points to aid in selecting the direction in which it should move. An activation-based matching scheme leads to significantly better results than its unbiased match predecessor that was developed earlier. As the agent acquires more memory items, the eater's performance continues to improve until it performs at a level comparable to the greedy eater (that only heads to the best food in its direct neighborhood) [18]. The hypotheses of cognitive capabilities resulting from this episodic memory are discussed and confirmed by implementations in their article [16].

LIDA - The Learning IDA (LIDA) architecture incorporates six major artificial intelligence software technologies: the copycat architecture, sparse distributed memory, pandemonium theory, the schema mechanism, the behavior net model, and the sub-sumption architecture [19]. LIDA is an extension for the Intelligent Distribution Agent (IDA) — which is a referred to as "conscious" software agent — by perceptual, episodic, and procedural learning capabilities. It was created as model of human cognition that could be used to suggest possible answers to questions about the human mind. The authors designed and developed a practical application that could act like a human detailer, a person who negotiates with sailors about new jobs who are near the end of their current tours of duty. A percept in the LIDA architecture can be thought of as a set of elements of a ontology that are relevant to the stimulus. They organize this information into a binary vector, where each field of one or more bits represents an element of the ontology [19]. A cue (the binary vector) will be used to query the content-addressable memories, autobiographical memory (ABM) and transient episodic memory (TEM). Both are based closely on Kanerva's sparse distributed memory (SDM) [20]. A similarity between SDM circuits and those of the cerebellar cortex are noted by [12]. Unfortunately, in this approach the whole domain must be specified within ontology. It is limited to the domain of providing new jobs to sailors.

Biological Models:

Make claims about the computation that support recall and recognition judgments, the main difference being that they also make specific claims about how the brain gives rise to these computations. While the former models account for challenging patterns of behavioral recall and recognition data from list learning paradigms, the brain-model mapping of the

latter models provides an extra source of constraints on the model's behavior. Even if Norman, Detre & Polyn [15] outline a comprehensive overview on computational models for episodic memory, only few robotic systems exist that make use of such models for learning.

2.3 Architecture and Patterns for EM:

2.4 Successful Episodic Memories types :

3 Implementation.

At this project there were at least three levels of implementation. Firstly the ***integration*** between WorldServer 3D environment and LIDA cognitive platform – the sensors and actuators exported through WS3D proxy interface were aligned with LIDA "virtual mind" so that creatures at WS3D environment passed to be agents controlled by LIDA.

After that, the environment was configured and implemented to correspond to the ***business requirements***: "There will be one creature at the arena center, turning around, observing events and taking note of them – using our basic episodic memory. This role is called "Detective". At the end of execution, the Detective will report, according to its observations, which jewels were collected, time of event and creature ID that collected it (by estimation). Also, it is not uncommon the situation of a creature to be stuck when passing by an object side. For these cases, detective shall be able to send a message to release the creature – this event also must be registered and reported. This report will demonstrate how episodic memory can be useful."

The third level was to design and implement a ***basic episodic memory*** that could be used by the "Detective" and attend to business requirements.

3.1 WorldServer 3D.

WorldServer3D is a 3D virtual environment developed at JAVA platform by Prof. Ricardo Gudwin's Research Group - ***cogsys***. In this virtual environment, you can create a set of virtual creatures, which are controlled by their sensors and actuators, managed through sockets. Recently a proxy interface was provided, which facilitates utilization.

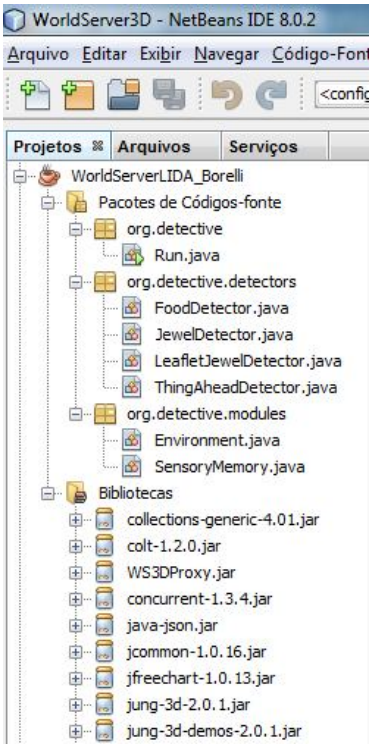
The main purpose of this environment is to provide a reference to the scientific community for analysis and tests of computational minds derived from various architectures and implementations.

The download of the WorldServer3D source code and its proxy interface can be done from the computer resources page of IA006 discipline:
http://faculty.dca.fee.unicamp.br/gudwin/courses/IA006/resources

3.2 LIDA – WS3D Integration.

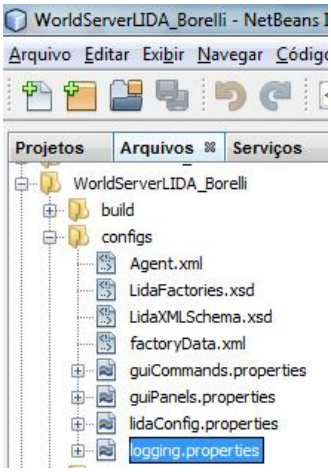
- LIDA Project creation:

As a first step, it was created a new Java project, WorldServerLIDA_Borelli, at NetBeans IDE. After that, selecting project properties - library option, all LIDA/lib .jar files were added to the Project, including lida-framework-v1.2b (LIDA root folder) and *WS3DProxy*.jar that provides communication interface to WorldServer3D:



- LIDA configuration files:

The configs folder was created with the following LIDA configuration files, as described at the tutorials and architecture:
Agent.xml, LidaFactories.xsd, LidaXMLSchema.xsd, factoryData.xml, guiCommands.properties, guiPanels.properties and lidaConfig.properties.



Based on the previous activities using *WS3DProxy* interface and LIDA exercises, the integration was developed. Three packages were created:

- org.detective: this package contains the main class.
- org.detective.detectors: contains the detection classes.
- org.detective.modules: contains classes related to the environment, creatures and sensorial memory.

The class org.detective.modules.**Environment** extends edu.memphis.ccrg.lida.environment.EnvironmentImpl that belongs to the LIDA library and represents the virtual environment for perception and control. The Environment class initialize all creatures and environment artifacts at the **init()** method:

```
public class Environment extends EnvironmentImpl {

    private static final int DEFAULT_TICKS_PER_RUN = 100;
    private int ticksPerRun;
   // private int stuckCounter =0;
    private WS3DProxy proxy;
    private Creature creature;
    private Creature detective;  //Detective - responsible to estimate
    private Thing food;
    private Thing jewel;
    private List<Thing> thingAhead;
    private Thing leafletJewel;
    private String currentAction;


    public Environment() {
        this.ticksPerRun = DEFAULT_TICKS_PER_RUN;
        this.proxy = new WS3DProxy();
        this.creature = null;
        this.detective = null;
        this.food = null;
        this.jewel = null;
        this.thingAhead = new ArrayList<>();
        this.leafletJewel = null;
        this.currentAction = "rotate";
    }
```

WorldServer 3D communication interface

```
public void init() {
    super.init();
    ticksPerRun = (Integer) getParam("environment.ticksPerRun", DEFAULT_TICKS_PER_RUN);
    taskSpawner.addTask(new BackgroundTask(ticksPerRun));
    try {
        creature = proxy.createCreature(100, 100, 0);
        creature.start();

        detective = proxy.createCreature(400, 400, 0);
        detective.start();

        World.grow(1);

        Thread.sleep(4000);
        detective.updateState();
        creature.updateState();

        System.out.println("Started...");
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Cria e inicializa o mundo virtual através da interface Proxy.

As oriented at the tutorial, the configuration files receive the class names for each specific module. The Environment class is indicated at the Agent.xml so that the LIDA platform can recognize the class that initialize the environment and communicate the events:

```
<module name="Environment">
    <class>org.detective.modules.Environment</class>
    <param name="environment.ticksPerRun" type="int">20</param>
    <taskspawner>defaultTS</taskspawner>
</module>
```

At the same way, detectors, sensory memory and other configurations are also declared at this xml file:

```
<module name="SensoryMemory">
    <class>org.detective.modules.SensoryMemory</class>
    <associatedmodule>Environment</associatedmodule>
    <taskspawner>defaultTS</taskspawner>
    <initialTasks>
        <task name="backgroundTask">
            <tasktype>SensoryMemoryBackgroundTask</tasktype>
            <ticksperrun>5</ticksperrun>
        </task>
    </initialTasks>
</module>
<module name="PerceptualAssociativeMemory">
    <class>edu.memphis.ccrg.lida.pam.PerceptualAssociativeMemoryImpl</class>
    <param name="pam.Upscale" type="double">.7 </param>
    <param name="pam.Downscale" type="double">.6 </param>
    <param name="pam.Selectivity" type="double">.5 </param>
    <param name="nodes">
        food,nearObject,leafletJewel,jewel
    </param>
    <taskspawner>defaultTS</taskspawner>
    <initialTasks>
        <task name="FoodDetector">
            <tasktype>FoodDetector</tasktype>
            <ticksperrun>3</ticksperrun>
            <param name="node" type="string">food</param>
        </task>
        <task name="ThingAheadDetector">
            <tasktype>ThingAheadDetector</tasktype>
```

The inner class ***private class BackgroundTask*** extends *lida.framework.tasks.FrameworkTaskImpl* that is responsible for agent operations at the virtual world.

```
private class BackgroundTask extends FrameworkTaskImpl {

    public BackgroundTask(int ticksPerRun) {
        super(ticksPerRun);
    }

    @Override
    protected void runThisFrameworkTask() {
        updateEnvironment();
        performAction(currentAction);
    }
}
```

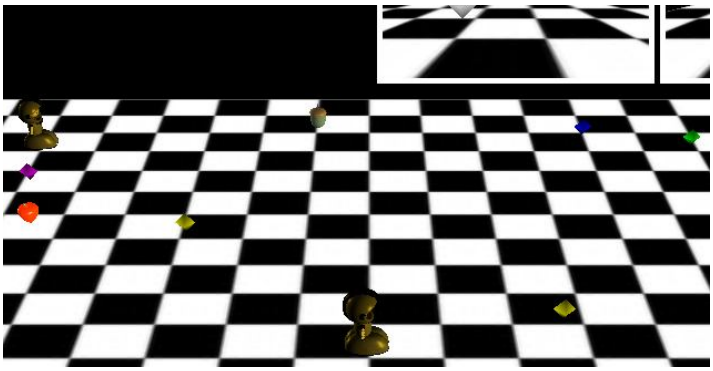
```
public void updateEnvironment() {

    // DETECTIVE:
    logic4Detectives();

    // CREATURES:
    logic4Creatures();
}
```

One behavior for detectives and other for common creatures.

The behavior for common creatures was defined according to its leaflet, energy and what is at its front. About the detective behavior, it has a detail – it will be turning around at the arena center and not eating. Then, at its behavior, we implemented Refuel tu be done every update.

3.3 The Detective Role and Basic Episodic Memory Utilization.

The figure below illustrates the WS3D arena with a creature at the upper left corner and the detective at the center. Both cameras are on and detecting.



The print below shows the detective camera detection. At this stage, it was possible to observe that the detective was able to detect the jewels and their positions, but not detecting the creature. Detective was wrongly detecting itself also. This was happening due the WS3D version.

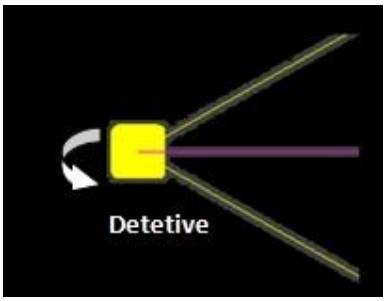
Action: rotate
Cor Jóia: Red
thing.getX1(): 761.5099282676305
thing.getY1(): 101.08824483809612
thing.getX2(): 773.5099282676305
thing.getY2(): 113.08824483809612
Cor Jóia: Green
thing.getX1(): 299.84325881153563
thing.getY1(): 36.73652300857365
thing.getX2(): 311.84325881153563
thing.getY2(): 48.73652300857365
Cor Jóia: Yellow
thing.getX1(): 587.3986882616115
thing.getY1(): 132.2377171039681
thing.getX2(): 599.3986882616115
thing.getY2(): 144.2377171039681
Cor Jóia: Magenta
thing.getX1(): 742.5034875886332
thing.getY1(): 56.80189971321468
thing.getX2(): 754.5034875886332
thing.getY2(): 68.80189971321468

After some running, the creature sack the green jewel, and the detection pass to be:

Cor Jóia: Red
thing.getX1(): 761.5099282676305
thing.getY1(): 101.08824483809612
thing.getX2(): 773.5099282676305
thing.getY2(): 113.08824483809612
Cor Jóia: Yellow
thing.getX1(): 587.3986882616115
thing.getY1(): 132.2377171039681
thing.getX2(): 599.3986882616115
thing.getY2(): 144.2377171039681
Cor Jóia: Magenta
thing.getX1(): 742.5034875886332
thing.getY1(): 56.80189971321468
thing.getX2(): 754.5034875886332
thing.getY2(): 68.80189971321468

Note that it is perfectly possible the detective to observe the missing jewel at that coordinate and registry the event and timestamp – this is the inspiration for the episodic memory to be dedicated for this case.

Self detection:
Nome Criatura: Creature_1436919018367
thing.getX1(): 380.0
thing.getY1(): 380.0
thing.getX2(): 420.0
thing.getY2(): 420.0 **this is the center position.**



The basic Episodic Memory was implemented using 3 Java HashMap classes. The **mapEpsdc** stores all jewel detected by the Detective during one turn.

The **mapEpsdcIN** keeps the new ones and **mapEpsdc** is cleared. The **mapEpsdcOUT** keeps all missing jewel that were, therefore, collected by the creature:

```
private void logic4Detectives() {
    detective.updateState();
```

```
  Set<String> KeysSET = mapEpsdc.keySet();
  TreeSet<String> sortedKeysSET = new TreeSet<String>(KeysSET);
for(String key : sortedKeysSET ){
    if (!mapEpsdcIN.containsKey(key)){
        mapEpsdcIN.put(key,mapEpsdc.get(key));
        ReportIN.append(key + "foi inserida na arena aproximandamente em:" + sdf.
    }
}
Set<String> KeysINSET = mapEpsdcIN.keySet();
  TreeSet<String> sortedKeysINSET = new TreeSet<String>(KeysINSET);
  for(String key : sortedKeysINSET ){
    if (!mapEpsdc.containsKey(key)){
        mapEpsdcOUT.put(key,mapEpsdcIN.get(key));
        mapEpsdcIN.remove(key);
        ReportOUT.append(key + "foi coletada pela criatura aproximadamente em:" +
    }
}
```

```
Relatório das jóias coletadas pela criatura - visão Detective:
Magenta_377.37099521763747_44.5849452925532l5foi coletada pela criatura aproximadamente em:jul 16,2015 15:25:12
Red_487.4131656587648_14.946712793233367foi coletada pela criatura aproximadamente em:jul 16,2015 15:25:12
Yellow_376.75341282941963_300.18911473435537foi coletada pela criatura aproximadamente em:jul 16,2015 15:25:13

Relatório das jóias observadas pelo Detective:
Blue_625.6786657783871_103.27546704651306foi inserida na arena aproximandamente em:jul 16,2015 15:25:12
Blue_636.9873568980383_101.72195257014663foi inserida na arena aproximandamente em:jul 16,2015 15:25:12
Magenta_377.37099521763747_44.5849452925532l5foi inserida na arena aproximandamente em:jul 16,2015 15:25:12
Red_487.4131656587648_14.946712793233367foi inserida na arena aproximandamente em:jul 16,2015 15:25:12
Yellow_277.8060362666665_103.10544148863633foi inserida na arena aproximandamente em:jul 16,2015 15:25:12
Yellow_376.75341282941963_300.18911473435537foi inserida na arena aproximandamente em:jul 16,2015 15:25:13
Magenta_377.37099521763747_44.5849452925532l5foi inserida na arena aproximandamente em:jul 16,2015 15:25:13
Red_487.4131656587648_14.946712793233367foi inserida na arena aproximandamente em:jul 16,2015 15:25:13
```

4 CONCLUSIONS.

The LIDA architecture seems to be very flexible and important for cognitive systems implementation.

It is clear the gain for the application and business in cases were episodic memory is implemented. At the present case, the Detective is capable, for example, to release the creatures that can be stuck.

The review turn clear the importance of learning and memory design to define the capabilities for the architecture.

Next step, as a compromise, this review will be complemented with more details about Episodic Memories and the new one implemented for LIDA .

REFERENCES:

[1] A.Newell, *Unified Theories of Cognition*: Harvard University Press, 1990.

[2] *Artificial General Intelligence*, 2008: Proceedings of the First AGI Conference

[3] J.L.McClelland, D.E. Rumelhart and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructures od Cognition*. Cambridge, MA: MIT Press, 1986.

[4] S. Franklin, *The LIDA architecture: Adding new modes of learning to an intelligent, autonomous, software agent.* In Proc. of the Int. Conf. on Integrated Design and Process Technology, San Diego, CA: Society for Design and Process Science, 2006.

[5] Johannes Engelkamp. *Das menschliche Gedachtnis.* Hogrefe, Verlag fur Psychologie, Gottingen, 1990.

[6] Endel Tulving. *Episodic and semantic memory.* In E. Tulving and W. Donaldson, editors, Organization of Memory, New York, 1972. Academic Press.

[7] Daniel L. Schacter and Endel Tulving. *Memory Systems 1994*. MIT Press, Cambridge, Sep. 1994.

[8] Endel Tulving and Hans J. Markowitsch. *Episodic and declarative memory: role of the hippocampus*. Hippocampus, 8(3):198–204, 1998.

[9] Endel Tulving. *Episodic memory: From mind to brain.* Annual Review of Psychology, 53(1):1–25, 2002.

[10] Jeff Hawkins. *On intelligence. Owl Books, New York, 2005.*

[11] Emilia I. Barakova and Tino Lourens. *Spatial navigation based on novelty mediated autobiographical memory.* In Mechanisms, Symbols, and Models Underlying Cognition, volume 3561/2005 of Lecture Notes in Computer Science, pages 356–365, Berlin / Heidelberg, 2005. Springer.

[12] Pentti Kanerva. *Sparse Distributed Memory*. MIT Press, Cambridge, MA, USA, 1988.

[13] Hans J. Markowitsch. *Neuropsychologie des Gedachtnisses.* Hogrefe Verlag fur Psychologie, Gottingen ¨ et. al, 1992.

[14] Michael C. Anderson, Kevin N. Ochsner, Brice Kuhl, Jeffrey Cooper, Elaine Robertson, Susan W. Gabrieli, Gary H. Glover, and John D. E. Gabrieli. *Neuralsystems underlying the suppression of unwanted memories*. Science, 303(5655):232–235, 2004.

[15] Kenneth A. Norman, Greg Detre, and Sean M. Polyn. *The Cambridge Handbook of Computational Cognitive Modeling, chapter Computational Models of Episodic Memory*. August 2006.