

CONAG: A REUSABLE FRAMEWORK FOR DEVELOPING “CONSCIOUS” SOFTWARE AGENTS

MYLES BOGNER, JONATHAN MALETIC, and STAN FRANKLIN

Division of Computer Science, Department of Mathematical Sciences

The University of Memphis

Memphis, TN 381520-3240

dr.myles.bogner@asynchrony.com, jmaletic@memphis.edu, stan.franklin@memphis.edu

Received (received date)

Revised (revised date)

ConAg is a reusable framework, written in Java, for creating “conscious” software agents. The system provides an application skeleton that can be customized by developers. Its particular focus is on these agents’ “consciousness” mechanism. A “conscious” software agent is a cognitive agent that integrates numerous artificial intelligence mechanisms to implement the global workspace theory, a psychological theory of mind. This article gives an overview of “conscious” software agents and it describes the architectural style of “conscious” software agents, from which ConAg is constructed.

Keywords: Agents, Cognitive Model, Reusable Framework

1. Motivation

For the past several years, the “Conscious” Software Research Group at The University of Memphis has been developing “conscious” software agents with the hopes of creating “smarter” software. From the onset, and continually more so as development progresses, it is clear that these agents are extremely complex and time-consuming to develop and implement. “Conscious” software agents are autonomous agents that range in functionality, from academic seminar organizers², to navy detailers responsible for naval personnel placement⁵. Autonomous agents are systems situated in and part of an environment⁴. They sense this environment and act on it over time, in pursuit of their own agenda, in such a way as to possi-

bly influence what they sense at a later time. Autonomous agents are coupled to their environment. In the most generic sense they include numerous animals such as humans along with software agents such as computer viruses and artificial life agents.

Many of these agents are also cognitive agents³. Cognitive agents are autonomous agents equipped with cognitive features such as concept formation, consciousness, emotions, long and short-term memory, meta-cognition, and perception. In this article, these cognitive features are used both in the folk-psychological and technical senses. “Conscious” software agents are cognitive agents that implement global workspace theory. Global workspace theory¹ is a cognitive theory of mind focusing on human consciousness. This theory postulates that consciousness provides for numerous functions, including adaptation and learning, decision making, and self-monitoring.

ConAg, the “Conscious” Agent Framework, provides many of these functions for autonomous agents implemented in software. ConAg is being developed to help facilitate these agents’ construction. It is a reusable software framework for building “consciousness” into software and serves as the backbone for the first implementation of global workspace theory (which is described in the next section). An architectural style for “conscious” software agents that defines ConAg is then described. The framework itself is presented, including its component-based structure. The framework’s use in two “conscious” software agents is lastly illustrated.

2. Global Workspace Theory

Global workspace theory is a cognitive model focusing on human consciousness. The theory postulates that human cognition is implemented by a multitude of relatively small, special purpose processes, almost always unconscious. Each process is autonomous and narrowly focused. It is very efficient, works at high speeds, and makes very few errors. Each process can act in parallel with others. This allows for the creation of a high capacity system such as the central nervous system.

In such a multi-agent system, coalitions of such processes, when aroused by novel or problematic situations, compete for access into a global workspace and, therefore, into consciousness. This limited capacity workspace serves to broadcast the coalition’s message to all the unconscious processes, in order to recruit other processes to join in handling and solving the current novel situation or problem. Consciousness provides for numerous functions such as identifying conflicts and prioritization.

It is helpful to think of processes going through five stages. First, unconscious processes, each working towards achieving a portion of an overall goal, form a coalition. A coalition is a set of processes that work together to perform a specific task. Second, coalitions compete for access into the global workspace. The global workspace provides a central location for one coalition to interact with the system’s other processes, similar to a blackboard. Third, items that enter the global

workspace, or the *spotlight of consciousness*, broadcast information to all unconscious processes. This broadcast allows the conscious coalition to recruit other processes which can contribute to the conscious coalition's tasks. The spotlight can only shine on one coalition at a time. For example, humans must think of two alternatives one after another; they cannot be addressed simultaneously. Fourth, all unconscious processes receive the broadcast message. Finally, the processes which understand this message, and which need to take action, do so.

3. An Architectural Style for “Conscious” Software Agents

This work involves the implementation of the global workspace theory to support the idea of “conscious” software agents for complex problem solving. The first such system developed is “Conscious” Mattie (CMattie)². At CMattie's onset, the plan was to apply these methods to a number of other problem domains. Therefore, it was decided to develop a reusable framework to support the implementation and development of such systems. The framework, ConAg, is developed in Java using object oriented concepts. ConAg's architecture rests on a software design philosophy that directs “conscious” software agents' development. The main goal was to identify the key design features common across application domains. This supports high level design reuse that in turn speeds the development of such systems.

In addition to adhering to global workspace theory, “conscious” software agents are designed following the action selection paradigm, a paradigm providing principles for a cognitive agent architecture³. This paradigm facilitates the realization of global workspace theory in software agents. Action selection states that minds are autonomous agent's control structures. A mind's task is to produce the next action. Minds should be viewed as continuous instead of Boolean. Sensations, such as perception, are operated on by minds to create information for their own use. A multitude of disparate mechanisms enable minds and there is little direct communication between them. Minds and action selection are limited to autonomous agents. Agents are situated in environments and the agents' actions are selected in the service of *drives*. Prior-information (memories) are re-created to help produce actions. Cognitive functions, as described by global workspace theory, such as categorizing, inferencing, planning, recalling, recognizing, and sensing all serve to help determine what to do next.

An architectural style for “conscious” software agents was defined to support the above mentioned concepts. This framework provided a collection of constraints, building-blocks, design elements, and rules for composing a “conscious” software agent¹⁰. This architectural style is given in Figure 1. It is analogous to the architectural style of a blackboard system, but extends those concepts to support global workspace theory and the action selection paradigm. These systems are composed of three main components: an attention manager, memory, and cognitive features. The attention manager gathers information together (using Pandemonium theory) and in turn updates the long and short term memory. The information in memory

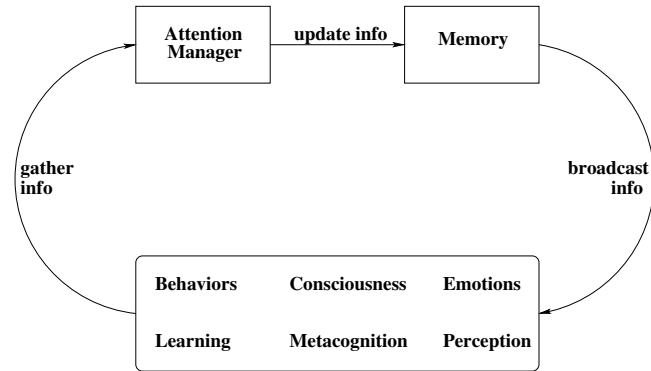


Fig. 1. Architectural Style For “Conscious” Software Agents

is continually broadcast back to the cognitive features.

Pandemonium theory’s⁶ components interact like people in a sports arena. Both the fans and players are known as *demons*. Demons can cause external actions, they can act on other internal demons, and they are involved in perception. The vast majority of demons make up the audience in the stands. There are also a small number of demons on the playing field. The ones on the playing field are attempting to excite the fans. Audience members respond in varying degrees to these attempts to excite them, with the more excited fans yelling louder. The loudest fan goes down on the playing field and joins the players, perhaps causing one of the players to return to the stands. The louder fans are those who are most closely linked to the players. There are some initial links in the system and more are created over time. Links are created and strengthened by the amount of time demons spend together on the playing field and by the system’s overall motivational level at the time.

By refining this architectural style to the next level, Figure 2 illustrates the particular architecture that is used in ConAg. This architecture’s modules are described below. ConAg is focused primarily on the items circled in this figure.

Many of the cognitive mechanisms such as behaviors and perception are, in reality, driven by small single-task *codelets* corresponding to global workspace theory’s processors and pandemonium theory’s demons. These correspond to the “Cognitive Process Codelets” of figure 2 and are the codelets that comprise emotions⁹, behaviors⁸, meta-cognition¹², and perception¹¹. Emotion codelets are dispersed throughout “conscious” software agents, looking for situations that will influence the systems’ overall emotional state. The system’s emotional states are a composite of several emotions, such as happiness, sadness, anger, and fear. Behaviors serve to perform the systems’ major actions. For example, agents that communicate via email may have a behavior to compose a reply to an incoming message. Working in conjunction with behaviors are drives. Drives are built-in to “conscious” software

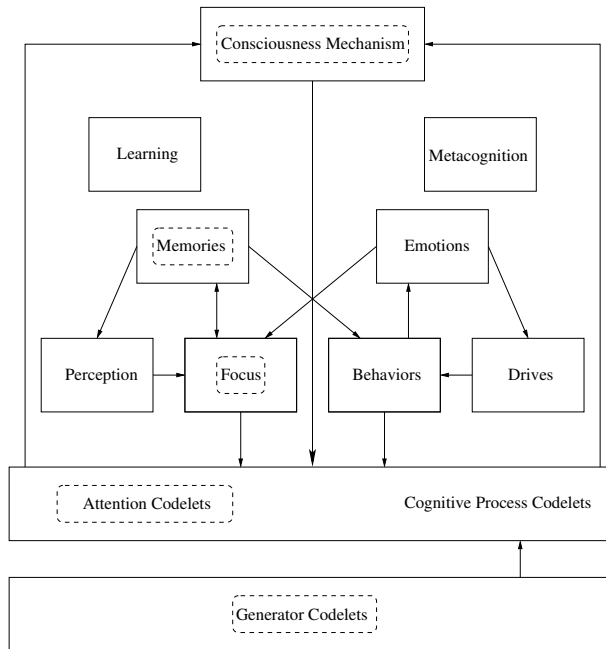


Fig. 2. An Architecture For “Conscious” Software Agents

agents and they operate in parallel. Drives activate behaviors, and behaviors work to fulfill them.

Another cognitive feature, perception, varies depending on the domain; it can range from receiving voice in tutoring systems to natural-language email messages in department seminar organizers. As seen in figure 2, the “Focus” is the location where this incoming perceptual information is created for the agent’s own use. Here, these perception are associated with emotions and various memories. Meta-cognition, as an overseer, monitors the agents’ internal conditions. If necessary, it can influence the behaviors, perception, “consciousness”, and learning. As an example, meta-cognition can make the agent more goal-oriented or opportunistic and cause voluntary attention by influencing the chances that a coalition of codelets will make it to “consciousness”. Learning works with meta-cognition and takes many forms in these agents such as the ability to learn new behaviors.

All codelets have activation levels corresponding to how important they perceive their action to currently be. These activation levels are also directly associated with the higher level concept the codelet serves, such as a behavior currently being executed. Codelets also contain associations with other codelets, corresponding to the links of pandemonium theory’s demons. All codelets that are actively performing their tasks join the playing field. The playing field is the first portion of the “conscious” attention mechanism. The primary responsibility of attention codelets is

to bring novel or conflicting information to “consciousness”². This includes new perceptual information and also includes conflicts between what is perceived and what is remembered along with conflicts in the potential communication output of the agents. The attention mechanism contains a way to form coalitions of codelets. Specifically, a coalition manager works to group codelets into coalitions based on their associations to other codelets. A coalition must be selected for “consciousness” from among the formed coalitions. This selection is performed by the spotlight controller and is based on a coalition’s activation level. Once the “conscious” coalition has been selected, the attention mechanism’s broadcast manager sends out the coalition’s information. Since it is known that human short-term memory holds approximately seven recent conscious items, the coalition’s information is placed in the system’s short term memory. All codelets in the system are able to receive these broadcasts. In some cases, there must be multiple copies of the same kind of codelets based on what is “conscious”. Generator codelets, each corresponding to a specific kind of codelet, solve this problem by receiving the broadcast and instantiating copies of themselves with the correct information.

“Conscious” software agents are often very domain-specific entities. Following the action selection paradigm, what an agent perceives, its drives and corresponding behaviors, etc. are coupled to its environment. A relatively domain-independent portion is the “consciousness” mechanism. This is ConAg’s main focus.

4. The “Conscious” Agent Framework

The idea of a reusable framework is often not well understood and the term is misused outside the object-oriented community⁷. Frameworks are reusable designs of all or part of a class of systems. They are commonly represented by a set of abstract classes along with the way the class instances interact. Frameworks are actual programs, or program shells. Their purpose is to provide an application skeleton that can be customized by developers. Frameworks are powerful because they significantly reduce the amount of effort necessary to develop customized applications, thereby saving time and money.

As a framework, ConAg serves four primary goals:

- (i) To fit within the boundaries of the architectural style for “conscious” software agents.
- (ii) To provide a drop-in implementation for the domain-independent portions of these agents’ “consciousness” mechanism.
- (iii) To provide working, easily customizable, and properly documented domain-specific portions of the “consciousness” mechanism, such as attention codelets.
- (iv) To provide stubs for the cognitive mechanisms such as emotions and behaviors found in “conscious” software agents.

ConAg is implemented in Java and utilizes Java’s AWT and Beans frameworks. All classes are implemented as Java Beans, where ever possible, helping contribute

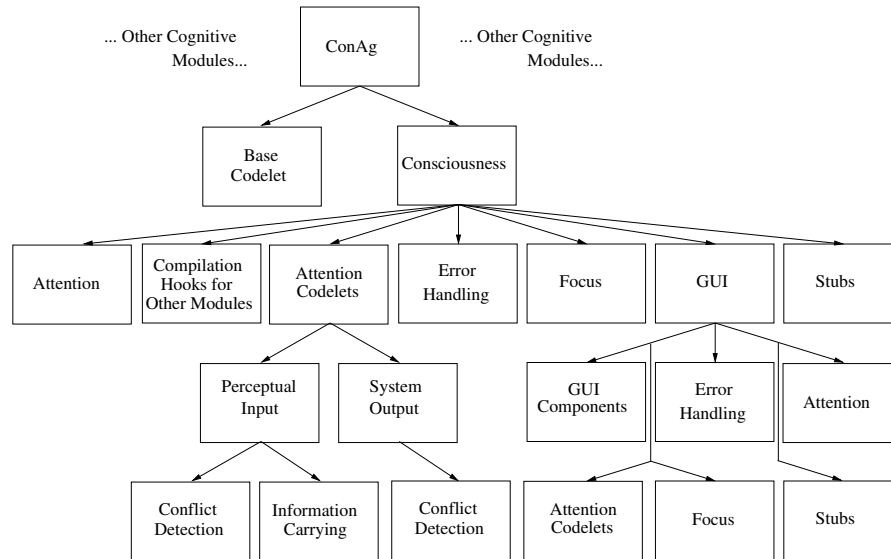


Fig. 3. ConAg’s Hierarchical Framework

to both black-box reuse and, when necessary, easy modification for white-box reuse. In fact, all components except for the attention codelets and Focus are designed to support white-box reuse. Each source file has detailed header comments, and almost every line of code is commented, with all comments catered for Javadoc use. In other words, explicit care has been taken to ensure ConAg follows good coding practice to support long term maintenance and reuse. ConAg has a detailed package structure allowing for components to be easily found and identified (see figure 3).

ConAg can be viewed partially as a generic framework similar to a framework for building graphical user interfaces. It implements portions of the “consciousness” mechanism intended to work across domains. In other words, many portions of the framework can simply be dropped in an agent being developed. As a framework, ConAg is placed in the hierarchy at the same level as frameworks for “conscious” software agents’ other main modules, such as emotion and meta-cognition. ConAg’s package branches off into two main groups, the base codelet and “consciousness” packages.

Within figure 3’s base codelet package, ConAg includes a definition for what a minimum codelet is. All codelets in “conscious” software agents must inherit this base codelet, customizing it for specific use. This is needed as codelets must contain certain properties to be appropriately recognized and handled by the attention package’s components described below. This codelet definition includes items such as: the way a codelet’s associations to other codelets is represented; the manner that

a codelet's activation level is represented; how the activation level is increased and decreased; the structure each codelet needs to carry in order for its information to be successfully broadcast; and the way a codelet receives the "conscious" broadcast and checks short-term memory.

In the attention codelets package, ConAg also includes a definition for a base attention codelet that all attention codelets inherit. This definition highlights that attention codelets carry both conflict and/or novel information. In addition, it contains the methods by which attention codelets rapidly increase their activation levels in their attempt to gain "consciousness." Pure black-box reuse of the attention codelets is difficult, as they are tied closely with the problem domain. The provided attention codelets therefore are typically altered to reflect these dependencies. ConAg provides attention codelets to detect novel and conflicting information for perceptual input and output. These codelets are catered for agents that communicate via email. As attention codelets are Java beans, modifying them for new problem domains does not always require changes to the graphical user interface.

The attention package includes a pandemonium-like playing field, a coalition manager, a spotlight controller for selecting the "conscious" coalition, and a broadcast manager. Because this system is based on pandemonium theory's concept of demons, a concept manager for learning new concepts is also included along with support for a short-term memory. While black-box reuse is intended for these components, each can be modified to the developer's satisfaction. For example, a new algorithm for forming coalitions can be created and easily integrated into the system.

As discussed previously, the Focus is the location where perceptual information is created for the agent's own use. ConAg's Focus package is currently tailored for email correspondence. White-box reuse is necessary to modify its structure to accept different forms of perceptual input. The Focus includes an event driven system that notifies attention codelets when new percepts and memories are placed within it. ConAg's stubs package supplies templates for other cognitive modules such as meta-cognition. These stubs are currently functional and provide the basic mechanisms for further development of other cognitive modules. Unlike the fully implemented cognitive modules, care has been taken to try and make these modules domain-independent and therefore very reusable.

ConAg's graphical user interface, found in the GUI package, is written using Java's AWT. It provides a window into the inner workings of the system. ConAg does not depend on the GUI to run and this package can be completely removed. Since the GUI is driven by Java beans, it is easily web enabled through Java Server Pages. The framework error handling package also provides a common mechanism for handling errors. The provided GUI contains its own error handling mechanism, using identical techniques as the framework's main one. This technique allows the developer a single point in which to code for handling additional errors while also providing consistent debugging methods throughout the framework.

5. Agents Using ConAg

ConAg has successfully been deployed in two experimental “conscious” software agents, CMattie² and IDA², with CMattie currently being the more mature of the two agents. CMattie is designed for a specific, narrow domain. She functions in an academic setting, gathering information from humans regarding seminars and seminar-like events such as colloquia, defenses of theses, etc. Using this information, she composes an announcement of the upcoming week’s seminars, and mails this announcement weekly to members of a mailing list that she maintains, again by email interactions with humans.

IDA, the Intelligent Distribution Agent, is a significant extension of CMattie. IDA is being designed and prototyped for the United State Navy⁵. IDA is designed to perform as a Navy detailer. At the end of a sailor’s tour of duty (approximately 3-6 years), the sailor is assigned to a new billet (job position). This job assignment process is known as distribution. These new assignments are made by approximately two hundred full-time navy personnel, known as detailers. Currently, employing these detailers costs approximately \$20,000,000 annually.

IDA is the “conscious” software research group’s proof of concept project. Like CMattie, IDA must communicate, this time with sailors, in natural language. In addition, she must access and understand the content of several naval databases. IDA has constraint satisfaction issues in satisfying the Navy’s needs. For example, she must make sure that a Naval destroyer has the required number of sonar technicians with the appropriate training. She must keep down the costs associated with moving sailors and she must cater to the desires and needs of sailors as much as possible.

In parallel to its continued development, ConAg is being thoroughly tested. The following is one of the many fine grained tests performed when ConAg is used in CMattie. The analysis asks three questions: 1) Do the Focus’ perception registers receive the data correctly from the perception module? 2) If so, do all the appropriate attention codelets pick up the information? 3) If this occurs, does all the information get chosen for “consciousness”?

To perform this test, ten different types of messages were chosen from email messages previously sent to the departmental secretaries. These message types ranged from requests to be added to a mailing list, to the addition of a new seminar, to announcements about an upcoming speaker talking on a specific topic in a certain seminar. These messages were placed in ConAg’s perception module stub. They were placed in the stub file in such a way as to mimic how CMattie’s perception module ideally performs. ConAg was run with these ten messages as input and then stopped. Upon completion, the output log file was analyzed to ensure that all of the perception registers were filled correctly for each message. Also, it was checked to make sure that all of the attention codelets that are supposed to pick the information did in fact spring into action. Finally, it was checked that the “conscious” broadcast contained the same information as the initial stub file.

Testing revealed that the perception registers were able to be set correctly 100% of the time. The attention codelets picked up their information 100% of the time and the data broadcast from “consciousness” matched the initially perceived information 100% of the time.

6. Conclusions

“Conscious” software agents continue to show great promise as systems that not only model the human mind but also solve challenging problems. These agents’ development is greatly facilitated by ConAg. With each success, “conscious” software agents gain additional complexities. ConAg’s grounding in design and code reuse allows the framework to continue to be enriched.

Acknowledgments

This work is supported in part by the Office of Naval Research (N00014-98-1-0332) and by the Department of the Navy (DAAH04-96-C-0086).

References

- [1] B. Baars, *A Cognitive Theory of Consciousness*, Cambridge University Press, New York (1988).
- [2] M. Bogner, *Realizing “consciousness” in Software Agents*, Ph.D. Thesis, The University of Memphis (1999).
- [3] S. Franklin, Autonomous Agents as Embodied AI, *Cybernetics and Systems* **28(6)** (1997) 499–517.
- [4] S. Franklin and A. Graesser, *Intelligent Agents III*, Chapter *Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents*, Springer-Verlag, Berlin (1997) 21–35.
- [5] S. Franklin, A. Kelemen, and L. McCauley, *Ida: A Cognitive Agent Architecture*, Proc. of the IEEE Conference on Systems, Man and Cybernetics (1998), 2646–2651.
- [6] J. Jackson, *Idea for a Mind*, SIGGART Newsletter **101** (1987) 23–26.
- [7] R. Johnson, *Frameworks = (Components + Patterns)*, CACM **40(10)** (1997) 39–42.
- [8] P. Maes, *How to do the Right Thing*, Connection Science (1990).
- [9] L. McCauley and S. Franklin, *An Architecture for Emotion*, AAAI Fall Symposium “Emotional and Intelligent: The Tangled Knot of Cognition” (1998).
- [10] R. Monroe, A. Kompanek, R. Melton, and D. Garlan, *Architectural Styles, Design Patterns, and Objects*, IEEE Software **14(1)** (1997) 43–52.
- [11] U. Ramamurthy, M. Bogner, and S. Franklin, “Conscious” Learning in an Adaptive Software Agent, Proc. of The Second Asia Pacific Conference on Simulated Evolution and Learning (SEAL98) (1998).
- [12] Z. Zhang, S. Franklin, and D. Dasgupta, *Metacognition in Software Agents Using Classifier Systems*, Proc. of AAAI98 (1998) 82–88.