SOAR: Mecanismos sub-simbólicos e de aprendizagem

Danilo Fernando Lucentini

***Abstract*—These article has the following proposes: split the modules of SOAR cognitive architecture in two groups: symbolic and sub-symbolic. Moreover, the SOAR modules of learning and sub-symbolic will be analyzed in more details and, finally, a discussion will be made on the positive and negative points in SOAR architecture.**

***Index Terms*— SOAR, symbolic, sub-symbolic, cognitive architecture, mechanism learning**

I. Introdução

A arquitetura cognitiva SOAR[1] visa criar uma arquitetura computacional de propósito geral que, basicamente, descreva como as pessoas pensam. Para tanto, esta tenta relatar da maneira mais precisa possível como os seres humanos resolvem problemas, aprendem novos conhecimentos e memorizam informações.

O SOAR foi inicialmente proposto por John Larid, Allen Newell e Paul Rosembloom (Laird, Newell, Rosembloom 1987) por volta de 1987. Inicialmente sua arquitetura era relativamente simples e com poucos módulos; porém, com o passar dos anos, novos módulos e funcionalidades foram integrados na arquitetura tornando-a muito mais robusta e diversificada. Atualmente, esta encontra-se na versão 9.3.1 (Julho de 2011) e é exatamente esta versão que será alvo de estudos desse trabalho.

Por volta dos anos 70 e 80, imperava na inteligência artificial o cognitivismo (também denominado GOFAI[2]). Essa vertente afirmava que muitos aspectos da inteligência podiam ser obtidos simplesmente pela manipulação de símbolos como defendido por Allen Newell e Herbet Simon (Newell, Simon, 1976). Símbolos são entidades que referem-se a objetos não por uma relação espaço-temporal, mas sim através de uma convenção, uma lei. Símbolos podem ser manipuladas para a geração de novos símbolos que terão novos significados e assim sucessivamente. Dize-se sub-simbólico todo processamento que não é simbólico, ou seja, que não possui claramente essa relação símbolo-objeto. Pode ser representado, por exemplo, por um conjunto de coeficientes numéricos em uma rede neural.

Devido também a contemporaneidade dos acontecimentos, o SOAR herdou muito desse período cognitivista e, na sua primeira versão, quase todo o seu processamento era puramente simbólico. Entretanto, com o passar dos anos, novos módulos foram inseridos na arquitetura tornando-a mais diversificada. Assim, esse trabalho tem como primeiro objetivo a divisão dos atuais módulos da arquitetura sob a óptica simbólica e sub-simbólica.

Além disso, outro ponto que este trabalho abordará diz respeito à aprendizagem. É fácil notar que aprendizagem e inteligência estão intimamente relacionados. Logo, se nosso objetivo é desenvolver agentes artificiais inteligentes, obviamente teremos que nos preocupar com quais processos de aprendizagem estes irão dispor. Portanto, o segundo objetivo desse trabalho é analisar também os processos de aprendizagem do SOAR (independentemente de serem simbólicos ou sub-simbólicos).

Finalmente, será feita uma análise das principais características, positivas e negativas, que a arquitetura fornece para quem quer começar a desenvolver agentes inteligentes baseado nesse framework.

II. Divisão modular da arquitetura

Em 2008, Laird destacou em seu trabalho (Larid, 2008) quais seriam os principais módulos e características das novas versões do SOAR como poder ser visualizado na figura a seguir.

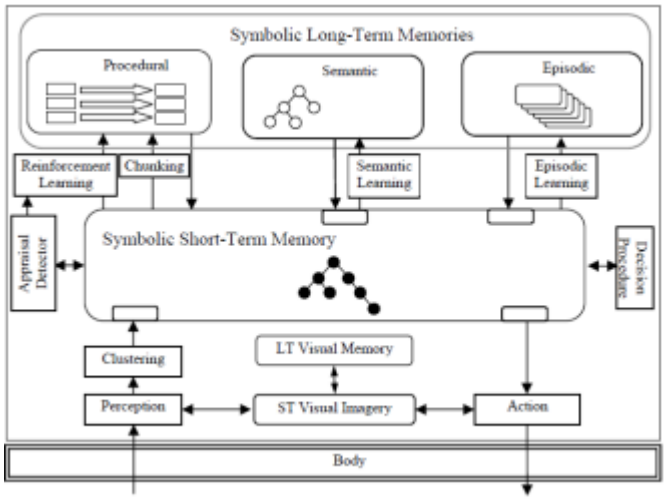

Ilustração 1: SOAR 9

Entretanto, muitos desses módulos ainda não estão integrados na versão 9.3.1 sendo sua implementação alvo de futuros estudos. Logo, para a atual versão, pode-se notar a existência real dos seguintes módulos: memória procedural, memória semântica, memória episódica, memória de curto-

[1]SOAR: State, Operator And Result

[2]GOFAI: Good Old-Fashioned Artificial Intelligence

prazo (memória de trabalho), aprendizagem por reforço, chunking, aprendizagem semântica, aprendizagem episódica, procedimento de decisão, percepção e ação. Os outros módulos, como dito anteriormente, não estão implementados na atual versão e, portanto, não serão abordados nesse trabalho.

Após a apresentação dos diversos módulos que constituem a arquitetura, partiremos para o primeiro objetivo desse trabalho: a catalogação de cada módulo segundo a óptica simbólica e sub-simbólica.

A. Memória de curto prazo (memória de trabalho)

É na memória de trabalho que residem os elementos básicos da arquitetura, os já conhecidos WME's[3]. Segundo o próprio manual do SOAR: "Cada WME contém um pedaço muito específico de informação (...) e vários WME's podem prover mais informação sobre o mesmo objeto contribuindo para a sua descrição" (Laird, Congdon, 2011). Obviamente, pode-se notar aqui um total relacionamento simbólico, pois cada WME está relacionado de alguma maneira ao objeto que este representa. Essa relação não é espaço-temporal e muito menos icônica, mas é dada através de uma convenção, uma lei (nos moldes da relação signo-objeto da filosofia Peirciana). Logo, esse módulo é puramente *simbólico*.

B. Memória semântica

A memória semântica é muito semelhante à memória de trabalho. Esta é constituída basicamente de WMEs cujos atributos são constantes (strings, integers ou decimals). Dessa maneira, como esse módulo é constituído fundamentalmente por WMEs, que poderão ser utilizados a longo prazo, é fácil observar que este módulo também é puramente *simbólico*.

C. Memória episódica

A memória episódica é um registro das experiências do agente (Laird, Congdon, 2011). Em suma, o seu conteúdo será dado pelo estado atual da memória de trabalho, ou seja, todo o conjunto de WME's (com atributos constantes) serão condensados e formarão um episódio. Novamente, nota-se que a memória episódica está intimamente relacionada aos WME's que já são simbólicos e, abstraindo ainda mais, cada conjunto de WME's está associado a um episódio que o agente vivenciou. Portanto, esse módulo também é *simbólico.*

D. Memória procedural

Este módulo contém todas as regras que determinam quando um operador deve ser escalonado para um determinado estado e quais as ações que este irá executar caso seja selecionado ao final do processo de decisão. Nota-se claramente uma relação entre as regras armazenadas e os operadores as quais estas se referem. Novamente essa relação é puramente dada por uma abstração, uma convenção. Assim sendo, também é possível inferir que a memória procedural é *simbólica*.

E. Aprendizado Semântico/Episódico

Estes dois módulos são independentes, mas como seus comportamentos são muito similares, eles serão tratados conjuntamente. Ambos os modelos são de aprendizagem e suas funções são: retirar um conjunto de dados da memória de trabalho e armazená-lo na respectiva memória (semântica ou episódica). É possível notar que somente ocorre um manuseio simbólico (símbolos da memória de trabalho são transformados em símbolos da respectiva memória), desta forma ambos os módulos são *simbólicos*.

F. Percepção

No SOAR, o módulo de percepção é basicamente uma interface que aguarda estruturas no formato de um WME. Nessa atual versão do SOAR, não existe nenhum mecanismo que gere um WME a partir de um conjunto de dados não simbólicos. Logo, esse módulo apenas recebe dados simbólicos e os repassa para a memória de trabalho, constituindo assim um sistema *simbólico.*

G. Ação

O módulo de ação segue os mesmos moldes do módulo de percepção, ou seja, é apenas uma interface onde pode ser "lido" um WME. Assim sendo, este módulo também é *simbólico.*

H. Chunking

O processo de chunking foi o primeiro módulo de aprendizagem do SOAR e está integrado na plataforma desde as suas primeiras versões.

Sua função é: criar novas regras que irão sintetizar um conhecimento que não é previamente sabido a partir do estado atual do problema. Seu funcionamento será mais bem explicitado nas próximas seções, mas já é possível inferir que como o procedimento visa criar novas regras a partir dos dados atualmente observados na memória de trabalho, então é possível inferir que também ocorre um manuseio de símbolos, pois os símbolos da memória de trabalho serão transformados em símbolos na memória procedural, portanto esse módulo também é *simbólico*.

I. Processo de Decisão

O processo de decisão do SOAR visa determinar qual operador será escolhido ao final de cada ciclo de decisão. Esse processo é determinado por meio de um conjunto de preferências e pelas condições atuais do problema.

Um operador é descrito, basicamente, por duas regras. Na primeira regra, o operador é proposto, ou seja, dado um conjunto de condições que devem ser satisfeitas, o operador é assinalado como plausível de seleção pelo SOAR (isto não garante, necessariamente, que o operador será executado). A segunda regra simplesmente relata o que o operador deve fazer, isto é, dado que o operador foi selecionado para execução, então são determinadas quais as ações que serão executadas.

Para que um operador possa ser executado, este deve ser plausível de execução (atenda ao estado atual do problema) e sua preferência deve ser maior que a de todos os outros operadores também plausíveis de execução.

[3]WME: Working Memory Element

Como a decisão sobre qual operador será executado é feita respeitando apenas critérios de preferência sem nenhuma relação a objetos ou símbolos, pode-se afirmar que o processo de decisão é *sub-simbólico.*

J. Aprendizagem por reforço

Tal módulo foi incorporado nas versões mais recentes do SOAR (Laird, Nason, 2004). Em suma, visa implementar a técnica clássica de aprendizagem por reforço na arquitetura.

Para as ações resultantes de um operador será atribuído um reforço (positivo ou negativo) e a partir desse reforço, nos próximos ciclos de decisão, este operador tenderá a ter uma preferência maior ou menor de execução quando comparado a outros operadores.

Portanto, esse sistema apenas regula preferências de operadores de acordo com um determinado reforço, o que faz dele um módulo *sub-simbólico.*

III. Módulos sub-simbólicos e de aprendizagem

Como visto no capítulo anterior, a maior parte dos módulos da arquitetura são simbólicos e uma pequena parte sub-simbólicos. Nesta seção será descrito, em maiores detalhes, o funcionamento dos módulos sub-simbólicos e de aprendizagem que compõem a arquitetura.

A. Processo de Decisão

O mecanismo de decisão é um dos módulos mais importantes e antigos da arquitetura estando presente desde o seus primórdios.

"O design do SOAR é baseado na hipótese de que todo comportamento deliberativo orientado a um objetivo possa ser traduzido com uma seleção e aplicação de operadores em um estado" (Laird, Congdon, 2011). Essa frase resume muito bem os principais alicerces do SOAR: *estado* e *operadores*.

O estado é a representação atual do problema e operadores são modificadores que realizam alguma transformação no estado atual.

Tudo no SOAR é baseado nessas duas entidades e qualquer modificação deve ser feita através de operadores. Dessa maneira, para um determinado problema, existem diversos operadores que podem ser propostos.

O módulo de procedimento de decisão tem como finalidade escolher quais são os operadores que podem ser aplicados a um determinado estado e qual, dentre os previamente selecionados, irá executar. Iremos denominar *ciclo de decisão* o ciclo que resume o funcionamento deste módulo, isto é, selecionar quais operadores estão aptos a serem executados e determinar qual o operador executará dentre aqueles que são plausíveis de execução. É importante frisar que apenas um único operador é selecionado ao final de cada ciclo de decisão. Assim, a mecânica do SOAR resume-se a ficar constantemente tentando localizar qual o operador que irá ser selecionando para um determinando estado calculando vários ciclos de decisões por minuto.

Agora iremos detalhar um pouco mais o funcionamento do ciclo de decisão.

Como dito anteriormente, cada operador é descrito, em sua grande maioria, por duas regras: uma que determina quando o operador será proposto e outra que determina quais as ações que o operador executará assim que este for selecionado. Logo, a primeira parte do ciclo de decisão busca quais operadores estão aptos a serem executados e simplesmente verifica quais regras são condizentes com o estado atual do problema.

Ao final da primeira etapa do ciclo de decisão, existirá um conjunto de operadores que estão aptos a serem executados, porém cabe ao módulo de decisão selecionar apenas um. Essa seleção é feita seguindo um critério de preferências, ou seja, se um operador tiver maior preferência que outro, o de maior preferência terá prioridade na escolha. Essas preferências são definidas via regra, isto é, junto com a definição dos operadores, é preciso definir quais são as prioridades desses operadores (globais ou relativas a outros operadores).

Porém, se o o módulo de decisão não conseguir encontrar um único operador para um determinado estado, ocorre um *impasse*

Um impasse nada mais é que um sub-estado no SOAR e o objetivo, naquele momento, torna-se resolver o impasse. Nesse estado estão contidas diversas informações como por exemplo: qual o tipo de impasse, quais e quantos são os operadores que entraram em impasse e outras informações.

Vale ressaltar que um impasse não é resultado de uma má elaboração de operadores e regras. Muito pelo contrário. O impasse é algo fundamental na arquitetura. Imagine que tenhamos o operador "tomar água" cuja regra de proposição seja "visualizar um copo de água". Se a visão da minha criatura detectar dois copos de água, dois operadores "tomar água" serão propostos e possivelmente ocorrerá um impasse. Graças a esse impasse é possível tomar alguma ação como, por exemplo, escolher o copo mais próximo. Esse é o poder do impasse: decidir em tempo de execução qual o melhor operador para uma determinada tarefa.

Os impasses são resolvidos através de regras que determinam qual será o único operador a ser executado. Então, para resolver o impasse anterior poderia haver uma regra do seguinte tipo: "se ocorrer um impasse entre os operadores 'tomar água', execute o operador que esteja referindo-se ao copo de água mais próximo".

Caso um sub-estado não seja resolvido, um novo sub-estado é gerado e assim sucessivamente até um limite ser atingido. Com isso é possível resolver impasses por partes, primeiro resolve-se um nicho do impasse e depois o nicho restante o que torna a arquitetura muito mais robusta.

B. Chunking

Chunking é o procedimento de criação de um novo conhecimento procedural (regras). O objetivo desse módulo é sumarizar o conhecimento aprendido anteriormente através de novas regras.

Como discutido na seção passada, quando um ocorre um impasse, um sub-estado é gerado. Esse sub-estado é por vezes

chamado de sub-objetivo, pois o objetivo naquele exato instante é resolver o impasse.

O chunking atua neste instante, isto é, caso o sub-objetivo tenha sido resolvido, este módulo irá tirar uma "foto" da memória de trabalho naquele exato momento e criará uma nova regra na memória procedural onde a "foto" tirada é a condição da nova regra e a ação tomada, a consequência da regra. Essa "foto" nada mais é que a memória de trabalho naquele momento sendo constituída de todos os seus WME's.

Desse modo é possível vislumbrar o principal objetivo do chunking: criar uma base procedural para os sub-objetivos, agilizando assim a tomada de decisão.

Entretanto, o uso desse módulo pode gerar alguns efeitos negativos. Caso o mecanismo de chunking crie um número excessivo de novas regras isso pode comprometer o desempenho do ciclo de decisão, pois será necessário a avaliação de uma maior quantidade de proposições (Kennedy, 2003) e, em particular, é necessário tomar muito cuidado com WME's que possuem um valor contínuo associado. Explicando: caso um sub-objetivo seja resolvido, o SOAR irá tirar uma "foto" da atual memória de trabalho e nessa foto estará contido o valor dessa WME de domínio contínuo. O problema é que caso o valor dessa WME mude, a regra criada pelo processo de chunking não será mais válida. Isso se agrava para WME's contínuas, pois existe uma infinidade de possibilidades, assim o processo de chunking só contribuirá para deixar mais lento cada ciclo de decisão. Dessa maneira, é necessário que se faça algum tipo de categorização desses dados contínuos de modo a eliminar tal problema.

C. Aprendizagem por reforço

Este é um dos módulos mais recentes a ser incorporado (Laird, Nason, 2004) e visa trazer para a arquitetura o mecanismo clássico de aprendizagem por reforço.

Inspirado na psicologia "behaviorista", o mecanismo de aprendizagem por reforço visa determinar qual a melhor ação a ser tomada dado que a cada ação executada um reforço, positivo ou negativo, é fornecido. Assim, cada ação estará associada a um valor que reflete a recompensa daquela ação. Logo, ações com recompensas positivas tenderão a serem executadas mais vezes que ações com recompensas negativas.

Dado que uma ação será executada no instante *t,* o cálculo da recompensa associada a essa ação neste instante varia de método para método, mas praticamente todos levam em consideração dois fatores: um dos fatores são as demais recompensas daquela ação nos instantes inferiores a *t* e o outro fator é o reforço obtido depois da execução da ação.

É notório perceber que esse mecanismo traz muitas vantagens como, por exemplo, a rápida adaptação em ambientes dinâmicos, independência de supervisão, entre outros.

No SOAR, podemos reescrever a frase "qual a melhor ação a ser executada" por "qual o melhor operador a ser executado". Para utilizar a aprendizagem por reforço no SOAR, operadores específicos devem ser utilizados que serão denominados operadores RL. A definição desses operadores é praticamente idêntica a dos demais operadores, exceto por uma característica: possui uma preferência de indiferença numérica. Um exemplo de um desses operadores pode ser visualizado a seguir:

```
sp {rl*3*12*left
    (state <s> ^name task-name
               ^x 3
               ^y 12
                ^operator <o> +)
   (<o> ^name move
        ^direction left)
-->
(<s> ^operator <o> = 1.5)}
```

A preferência de indiferença numérica é exatamente a última linha do operador (= 1.5).

Os operadores RL funcionam como qualquer outro operador no SOAR sujeitos, inclusive, a impasses com os demais operadores. Aqui pode-se perceber uma separação clara de responsabilidades: o módulo de aprendizagem por reforço visa determinar quais as preferências dos operadores RL, já a escolha do operador que irá realmente ser executado fica a cargo do mecanismo de decisão e segue os mesmos critérios já relatados.

O reforço associado a cada operador fica armazenado em um estrutura na memória de trabalho (uma WME). O SOAR irá checar essa estrutura em busca de um número que represente o reforço associado ao último operador executado no começo de cada ciclo de decisão.

Com o reforço em mãos, o módulo consegue atualizar as preferências dos operadores RL, assim operadores com um reforço positivo tenderão a ter maior preferência. Essa atualização de preferências pode ser realizada através de dois algoritmos: SARSA ou Q-Learning e pode ser melhor descrita em (Laird, Congdon, 2011).

D.Aprendizado Semântico/Episódico

Como dito anteriormente, esses módulos são independentes porém seus comportamentos são tão similares que serão tratados em conjunto com as devidas distinções.

Ambos os módulos tem como finalidade principal armazenar um valor ou um conjunto de valores da memória de trabalho para a memória semântica ou episódica, porém essa armazenagem é feita de maneira diferente por cada um dos módulos.

A memória episódica, como dito anteriormente, é semelhante a um conjunto de "fotos" da memória de trabalho. Cada "foto" ou episódio (como é mais comumente chamado) corresponde à situação exata da memória de trabalho naquele instante contendo todos os respectivos WME's. Por isso diz-se que a memória episódica é altamente contextualizada.

O armazenamento de dados na memória episódica é feito automaticamente sem requerer nenhuma ação deliberada por parte do agente, mas o instante e a frequência com que são armazenados novos episódios podem ser parametrizados pelo

sistema.

Já a memória semântica é dita independente de contexto, pois não são salvos todas os WME's de um determinando instante. Ao invés disso, o agente deve explicitar claramente qual WME deve ser salvo. Por isso a memória semântica é independente de contexto, pois esta tende a representar um conjunto de conhecimento mais geral.

Além da armazenagem em si, um ponto muito importante é: como recuperar os dados já armazenados? Neste quesito os dois módulos também apresentam algumas diferenças e semelhanças entre si.

Comecemos pelas semelhanças: a recuperação de dados já armazenados é feita via *cue* em ambos os casos. Pode-se fazer uma analogia de cue com uma query em SQL. Basicamente, deve-se montar uma cue que busque na memória semântica ou na episódica um conjunto de WME's com certas características da mesma maneira que uma query em SQL busca no banco de dados um conjunto de informações. Além disso, as buscas são assíncronas, isto é, após uma cue ser criada e disparada, o retorno dos dados será colocado em um WME específico assim que a busca tiver sido finalizado e não necessariamente de maneira subsequente à requisição.

Essa assincronia pode parecer estranha a primeira vista, mas vale ressaltar que "por baixo dos panos" esse mecanismo é implementado através de um banco de dados SQLite e uma busca em banco de dados pode demorar um tempo razoável dependendo do tamanho do banco ou do tipo de busca a ser realizada. Logo, não parece razoável o SOAR ter que "permanecer parado" aguardando o resultado da busca. Assim, para utilizar esses tipo de estrutura é necessário que haja um operador que crie a cue e a dispare e outro operador que fique verificando constantemente uma WME específica que indica se o resultado da busca já retornou.

Contudo, existem certas diferenças em como é feita a busca de dados nas estruturas. Na memória semântica essa busca é feita de maneira exata, assim todos os dados da cue devem corresponder exatamente aos dados retornados pela busca.

Já na memória episódica essa busca pode ser imaginada como uma sendo na vizinhança mais próxima e é feita em duas etapas: primeiro calcula-se quais os episódios que tenham pelo menos uma WME em comum com a cue. Em seguida, para esses episódios previamente selecionados, um cálculo é feito levando-se em consideração a cardinalidade do episódio (número de WMEs que satisfazem o episódio e a cue) e a ativação do episódio (a grosso modo pode ser imaginado como o nível de esquecimento do episódio, quanto maior a ativação, mais recente é o episódio). Assim, serão retornados os episódios que obtiverem as melhores notas. Para maiores informações de como são calculadas essas notas consulte (Laird, Nuxoll, 2007) e (Laird, Congdon, 2011).

IV. Discussão

Após a analisa dos vários módulos que compõem a arquitetura, pode-se notar que o SOAR ainda continua sendo uma arquitetura quase puramente simbólica, porém tudo leva a crer que será cada vez mais comum a implantação de novos módulos sub-simbólicos, refletindo uma tendência já incorporada em outras arquiteturas cognitivas, como por exemplo a arquitetura CLARION[4], que provê um hibridismo entre mecanismos simbólicos e sub-simbólicos a fim de obter resultados mais significativos.

Outro ponto que merece uma discussão mais profunda é com relação aos benefícios e limitações que a arquitetura proporciona.

Certamente um ponto muito positivo dessa arquitetura quando comparada a outras é a sua evolução, documentação e consolidação. O SOAR remonta de 1987 e de lá para cá tem sido mantido em constante atualização, inclusive com a criação de novos módulos que cada vez mais reproduzem comportamentos inteligentes em criaturas artificiais

Outro ponto positivo é a sua independência de plataforma. Quando o comportamento de um agente artificial passa a ser descrito através do SOAR, a propagação desse comportamento para outros agentes tende a ser mais simples, pois o comportamento é descrito através do SOAR e pode ser replicado, com certa facilidade, para diversos outros agentes. A ideia é muito similar ao que acontece em certas linguagens de programação, como o Java. O código não é compilado para uma plataforma em específica; mas, ao invés disso, compilado em *bytecodes* Java. Assim, basta criarmos a apropriada "virtual machine do SOAR" para uma determinada plataforma que o comportamento pode ser replicado com certa facilidade.

Porém o SOAR ainda possui uma série de limitações que ainda entravam a disseminação do uso dessa plataforma. Podemos destacar:

- Não é possível utilizar funções matemáticas na condição de uma regra. Isso gera um grande entrave em certas ocasiões pois é necessário criar um operador que realize a função matemática, porém o valor calculado pode não ser mais válido no próximo ciclo de decisão.
- Precisão numérica limitada a float e a falta de outros tipos de dados (como data e hora, por exemplo)
- Memória Episódica: não é possível atualizar ou remover dados
- Memória Semântica: não é possível remover dados
- Aplicações em tempo real. Devido ao mecanismo de resolução de impasse, não é possível garantir que o SOAR sempre gerará uma resposta no mesmo período o que gera um grande entrave para aplicações em tempo real.
- Criação de algum tipo de chunking customizável que possibilite a exclusão de WME's não relevantes para a criação de uma nova regra procedural, a fim de evitar o problema em espaços contínuos já relatado.
- Um único operador executado a cada ciclo de decisão. Estendendo a mesma ideia elaborada por

[4]CLARION: Connectionist Learning with Adaptive Rule Induction Online

(Dorer, 2004), cada operador poderia estar associado a um conjunto de atuadores. Assim, caso exista mais de um operador apto à execução e estes não comutam de atuadores em comum, tais operadores\ podem ser executados paralelamente.

V. Conclusão

É possível concluir que o SOAR apresenta bons instrumentos para o desenvolvimento de agentes inteligentes artificiais sendo assim uma excelente arquitetura cognitiva para propósitos gerais.

Além disso, é importante frisar que, apesar de suas origens behavioristas, o SOAR está tendendo a tornar-se uma arquitetura híbrida com módulos simbólicos e sub-simbólicos.

Finalmente, foi possível constatar que ainda existem muitos entraves que não permitem uma maior disseminação da arquitetura, porém diversos avanços já foram efeitos desde a sua criação e, com o passar dos anos, mais e mais pesquisadores tem-se unido para tornar o SOAR uma arquitetura cognitiva que realmente consiga criar comportamentos inteligentes em entidades artificiais

Referências

[1] Laird, J., Newell, A., and Rosenbloom, P. (1987). SOAR: An architecture for general intelligence. Artificial Intelligence, 33, 1-64.

[2] Newell, Allen; Simon, H. A. (1976), Computer Science as Empirical Inquiry: Symbols and Search, "Communications of the ACM", Communications of the ACM 19 (3): 113–126

[3] Laird, J. E. (2008). "Extending the Soar cognitive architecture," in Artificial General Intelligence 2008: Proceedings of the First AGI Conference.

[4] Laird, J.E., Congdon C.E. (2011). “The SOAR User's Manual Version 9.3.1”

[5] Laird, J.E., Nason, Shelley (2004). “Soar-RL: Integrating Reinforcement Learning with Soar” in International Conference on Cognitive Modeling.

[6] Kennedy, W.G (2003)., Manually Excising Soar Chunks During Long-term Learning

[7] Laird, J.E, Nuxoll, A.M (2007), Extending Cognitive Architecture with Episodic Memory .

[8] Dorer, K. (2004). Extended Behavior Network s for Behavior Selection in Dynamic and Continuous Domains.