INTEGRANDO SOFTWARE E HARDWARE EM AGENTES INTELIGENTES

Vítor Luís Babireski Fúrio[1] e **Maiquel de Brito**[2]

[1,2]Universidade Federal de Santa Catarina — Blumenau, SC, Brasil [1]Bolsista PIBIC – Propesq/UFSC/CNPq

Agente - um conjunto de hardware e software

Na área de inteligência artificial distribuída, agentes autônomos são usualmente tratados exclusivamente como elementos de software. Essa abordagem tem se mostrado suficiente para a execução de simulações e atuação em ambientes virtuais. Entretanto, quando se tem interesse em atuar em sistemas ciberfísicos, esta abordagem trata o agente como um "operário" que comanda a máquina física. Dessa forma, sensores e atuadores são tratados como elementos externos ao agente.

A proposta deste trabalho é apresentar uma abordagem diferente para aquilo que chamamos de *agentes ciberfísicos*. Nesta concepção, agentes são entidades que integram hardware e software e são capazes de atuar em diversos ambientes físicos, exercendo suas características de autonomia, pró-atividade, criatividade e sociabilidade. Estes agentes serão definidos pelas ações que podem realizar no ambiente físico, chamadas de *ações intrínsecas* e pelas crenças que são capazes de obter deste mesmo ambiente. Esta abordagem é implementada através de uma extensão à linguagem de programação de agentes *Jason*.

Motorista Vs carro autônomo

Para ilustrar a ideia, imaginemos o exemplo de um carro. Na abordagem tradicional, em que agentes são apenas programas de computador, o agente teria meios de controlar os sistemas do carro. Entretanto, agente e veículo seriam entes separados, semelhante a um motorista que dirige seu automóvel. O motorista não faz parte do carro, mas ele, assim como o agente, tem meios para controlá-lo. Por exemplo, é possível acelerar ao pisar no pedal do acelerador, checar o ambiente traseiro via espelho retrovisor, checar a gasolina via luzes do painel, frear ao acionar as alavancas de freio, etc.

O motorista usa de elementos externos a ele (pedais, painéis, espelhos) para conseguir realizar as ações que deseja. Algo semelhante acontece na programação orientada a agentes em geral, em que o agente usa artefatos externos a ele para controlar os sensores e atuadores. Na abordagem deste trabalho, propomos tratar agente e veículo como uma entidade única, de forma que as ações que são intrínsecas ao agente sejam programadas em seu próprio código (Java) e as crenças sejam adquiridas sem a necessidade de invocar funções para verificar valores dos sensores. No exemplo do carro, o agente novo pode ser definido como um carro autônomo cujas ações intrínsecas são acelerar, frear, virar, etc. Ele também apresenta a capacidade de obter, naturalmente, crenças como a quantidade de gasolina no tanque ou se há obstáculos ao redor.

Ver sem “ter” que observar

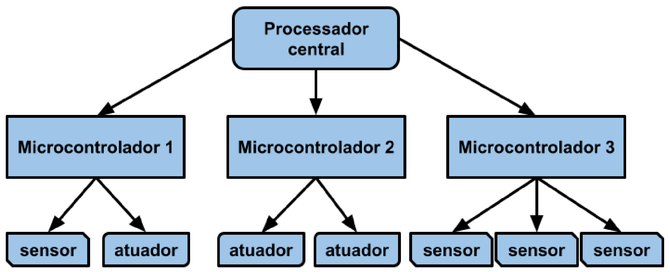
A ideia desta arquitetura é que as crenças venham “de forma natural” até o estado mental do agente. Com isso, queremos dizer que não há necessidade de incluir instruções para obtenção de crenças dentro dos planos de ação dos agentes (.asl). A própria arquitetura fará a checagem dos sensores a cada ciclo de raciocínio do agente e incorporará esses valores à sua base de crenças.

Esta abordagem baseia-se na forma como os humanos obtêm as informações sobre o seu ambiente, via sentidos. Por exemplo, não precisamos nos concentrar em ver para conseguirmos usar os nossos olhos e observar ao redor. Simplesmente fazermos isso, constante e involuntariamente. Esta mudança visa tornar a escrita do plano do agente mais simples e funcional, uma vez que todas as crenças que o agente pode obter de seu ambiente estarão disponíveis e atualizadas em seu estado mental.

A arquitetura

De forma geral, esta arquitetura visa permitir a programação de um agente que represente um robô. Este robô tem um **Processador Central**, com recursos suficientes para a execução de um agente, que pode controlar diversos **Microcontroladores**. Estes, por sua vez, são responsáveis pelo comando dos componentes eletrônicos (**sensores** e **atuadores**) a eles conectados. Um exemplo da hierarquia das camadas de hardware é mostrado abaixo:

Exemplo das camadas de HARDWARE



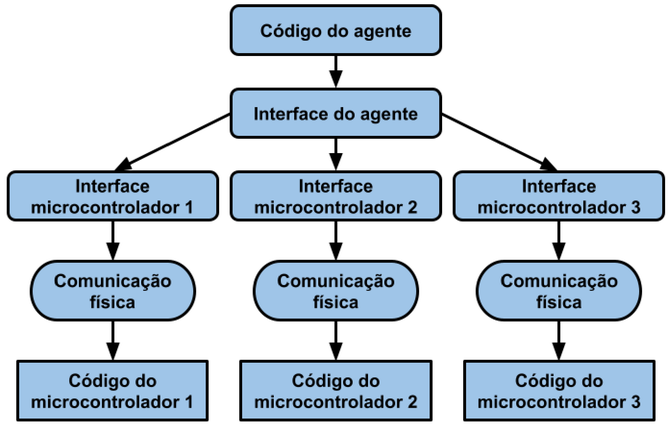
Os microcontroladores comandam os componentes eletrônicos através do **Código do microcontrolador**. Existe uma camada que garante a **Comunicação física** entre microcontrolador e processador central, assegurando a entrega dos valores lidos pelos sensores e das mensagens de ativação dos atuadores.

A **Interface do microcontrolador** é uma camada de código que representa o microcontrolador dentro do processador central. Nela o programador deve informar quais e quantos são os microcontroladores, bem como configurar suas conexões com o processador central.

A **Interface do agente** é a camada mais importante desta arquitetura. Nela, encontram-se as principais diferenças propostas neste trabalho. Aqui, o desenvolvedor irá especificar quais crenças o robô é capaz de adquirir através dos sensores e quais são as *ações intrínsecas* que o hardware é capaz de realizar. Esta camada é responsável por tornar essas funcionalidades (ações intrínsecas e crenças atualizadas) disponíveis em alto nível para o código do agente.

Por fim, para definir o comportamento do robô, o programador irá escrever o **Código do agente** em *Jason*. Um exemplo da hierarquia das camadas de software pode ser visto na figura abaixo:

Exemplo Camadas de SOFTWARE



A diferença prática

Um exemplo que mostra como essa abordagem se difere da tradicional em termos práticos é o exemplo do braço robótico. Um braço robótico é uma ferramenta extremamente versátil na indústria, podendo ser programado para realizar as mais diversas tarefas, como acionar botões, mover objetos de lugar ou utilizar ferramentas para corte e solda. Entretanto, existem funcionalidades que são intrínsecas desta ferramenta, como a capacidade de agarrar e rotacionar. Independente da tarefa que o braço realize, essas ações serão utilizadas, pois fazem parte do que define um braço robótico. Assim, quando pensamos que nosso braço robótico é um agente, essas ações também são utilizadas para defini-lo.

Desta forma, este trabalho visa desenvolver uma arquitetura que consiga representar essas *ações intrínsecas* do agente (no próprio código que define o que é um agente - Java). Para isso, estamos estendendo o interpretador da linguagem de programação de agentes *Jason* de forma que seja possível realizar essa representação. Assim, um projetista pode usar essa extensão para definir as *ações intrínsecas* do seu agente ciberfísico. Uma vez que essas ações estão definidas, elas são facilmente referenciadas durante a escrita do código do agente (.asl) de forma que, para uma mudança comportamental do agente (por exemplo, o braço robótico recebeu uma nova tarefa, ao invés de apertar botões ele vai utilizar uma máquina de corte), basta realizar alterações no código de agente (.asl), definindo como ele cumprirá seus novos objetivos usando as suas *ações intrínsecas*, sem a necessidade de alterar a porção de software que trata da manipulação do hardware.

A (re)utilização direta de código

O fato de o controle do hardware não ser feito no código do agente evidencia uma vantagem dessa abordagem: permitir que o código de agente seja reutilizado. No caso de dois agentes que apresentam diferenças em seus hardwares, no entanto, ambos realizem as mesmas funções e apresentem o mesmo comportamento, torna-se possível utilizar o mesmo código do agente para as duas entidades, uma vez que esta arquitetura visa permitir essa flexibilidade.

Levando esta idéia mais adiante, poderíamos ainda programar e testar o comportamento de um robô em um ambiente virtual e, em seguida, migrar o código de agente para dentro da entidade física, contanto que as funcionalidades sejam compatíveis, ou seja, desde que as *ações intrínsecas* do robô e as crenças que ele pode obter do ambiente sejam as mesmas existentes na simulação.

Andamento do projeto

O trabalho em questão encontra-se em fase intermediária de desenvolvimento. É possível acompanhar o seu progresso visitando a página do GitHub `https://github.com/embedded-mas/embedded-mas` ou entrando em contato com os autores pelos seguintes endereços de e-mail: [2] maiquel.b@ufsc.br, [1] vitor.furio@grad.ufsc.br ou [1] vitorluis.babireskifurio@gmail.com

Agradecimentos

Os autores gostariam de agradecer ao Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq – Brasil pelo apoio e incentivo neste projeto que foi contemplado com uma bolsa de iniciação científica.