Behavior Editor for Agents Based on Service Oriented Architecture

Saulo Popov Zambiasi
Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina
Florianópolis, SC – Brasil
saulopz@gmail.com

Ricardo J. Rabelo
Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina
Florianópolis, SC – Brasil
rabelo@das.ufsc.br

***Abstract*—Many efforts have been made to create agents more autonomous, flexible and dynamic. Concomitantly with the fact that agents are inserted in environments distributed, interconnected and with a lot of dynamics information, it agents need to communicates with others and other systems by the inter-operable way. The use of a standard, such as Service Oriented Architecture and web services, can provide an inter-operable way to agents communication and to executes some distributed operations. Thus, this paper presents a proposal of a behavior editor for agents based on Service Oriented Architecture. A prototype of a behavior editor and an executor of agents was implemented and it is also presented.**

Keywords—agents; behavior; service-oriented architecture.

I. INTRODUÇÃO

O cenário da computação distribuída tem aumentado significativamente a complexidade dos ambientes computacionais. Consequentemente, as pessoas veem-se inseridas em um ambiente pessoal cada vez mais distribuído e com uma grande quantidade de informações altamente dinâmicas. Dentro desse contexto, os agentes, como forma de auxiliar seus usuários, precisam interagir com este ambiente e manter a compatibilidade com os mais diversos tipos de conteúdos espalhados neste meio [2]. Para isso, técnicas para a criação de agentes e algoritmos para compor a forma como os agentes agem têm sido estudadas e aperfeiçoadas.

Uma maneira para resolver problemas de interoperabilidade nesse cenário distribuído é a utilização de estruturas e protocolos padronizados. A utilização de serviços distribuídos pelos agentes, via Arquitetura Orientada a Serviços (*Software Oriented Architecture* – SOA), traz consigo as vantagens dessa tecnologia. Nessa, as funções dos agentes se apresentam como serviços independentes, cada qual com interfaces de invocação bem definidas e que podem ser utilizadas sequencialmente ou paralelamente para compor processos de negócios [8].

Enquanto SOA é vista como uma forma bastante eficiente de executar programas remotamente via Internet, utilizando protocolos amplamente utilizados na atualidade, os agentes são uma tecnologia já firmada na computação como uma maneira para resolver problemas complexos [4]. Com a utilização de SOA em agentes, é possível que um agente possa se utilizar de chamadas à serviços web para efetuar algumas operações, ou mesmo processos de negócios. Porém, o usuário precisa poder personalizar as ações do seu agente, adicionar chamadas à serviços web, formas de interação, etc. de modo a este agente possa se adaptar às suas necessidades [9] e sem que o mesmo precise ser recompilado. Nesse contexto, esse artigo apresenta uma proposta de edição e execução de comportamentos de agentes integrados com SOA. Também é apresentado um protótipo implementado para os testes da proposta.

II. BREVE REVISÃO DA LITERATURA

Um agente computacional é visto como uma entidade que: (i) tem uma identidade persistente para realizar transações, manipular exceções, construir uma história de interações e definir a confiança para com outros agentes; (ii) percebe e responde ativamente às suas atividades no seu ambiente; (iii) se comunica com outros agentes ou pessoas [13]. Os agentes devem refletir a mesma dinâmica que um usuário apresenta na vida real, ou seja, assim como os interesses do usuário mudam com o tempo, os agentes também devem ajustar seus objetivos para corresponder com seus usuários [12].

Uma importante questão dos agentes está em construí-los de forma que possam ser facilmente personalizados para cada usuário. Muitos programas proveem simples parâmetros que permitem aos usuários personalizarem seus comportamentos explicitamente. Porém, em geral, essa interação é bastante limitada. Uma maneira sofisticada de se resolver esta limitação é por meio da dinamicidade e da escalabilidade. Com isso, muitos usuários podem ter disponível a possibilidade de programar suas preferências de forma explícita. Ainda assim, algumas tarefas, como personalizar um filtro de e-mails para selecionar mensagens urgentes, por exemplo, necessitam que a pessoa tenha certa noção detalhada e possa articular conceitos bastante sutis. Uma forma bastante interessante para resolução desse problema é deixar que especialistas desenvolvam os filtros e ao usuário fica apenas o papel de utilizá-lo e configurá-lo conforme suas necessidades. Entretanto, o problema de se lidar com a flexibilidade da forma como o agente se comporta vai além de simplesmente alterar a configuração de um agente para cada usuário [9].

Observa-se também a necessidade da utilização de padrões para comunicação entre agentes e outros softwares [2]. A melhor forma de se resolver esse problema é adotar algum tipo de tecnologia já difundida, tal como SOA, por exemplo.

SOA *"é um paradigma para organização e utilização de competências distribuídas que estão sob o controle de diferentes domínios proprietários"* [3]. As pessoas e organizações criam competências para resolver problemas específicos conforme suas necessidades, modeladas por um

conjunto de serviços de softwares [1]. Esses serviços possuem interfaces que podem ser publicadas e descobertas por consumidores de serviços e podem ser agrupados para a criação de diferentes aplicações e processos de negócios, utilizando-se de um modelo de comunicação baseado na troca de mensagens com baixo acoplamento [6], [10].

Os serviços web, por sua vez, são softwares que buscam a interoperabilidade através da interação entre um computador e uma rede. Estes serviços se comunicam entre eles e entre sistemas via mensagens baseadas no protocolo HTTP (*Hipertext Trasnfer Protocol*) [5].

Uma das vantagens de se trabalhar com serviços web em SOA está na estrutura flexível que permite a criação de novos serviços com a composição de outros serviços. Além disso, eles seguem os requisitos de SOA: (i) **Baixo acoplamento**, os serviços da arquitetura não devem ter uma dependência forte entre si; (ii) **Independência de implementação**, não se deve depender de características específicas de linguagens de programação ou ambientes de execução; (iii) **Configuração flexível**, os diferentes serviços devem poder ser ligados entre si de forma dinâmica e configurável; (iv) **Tempo de vida longo**, os serviços devem existir por tempo suficiente para que sejam descobertos e utilizados até se obter confiança em seu comportamento; (v) **Granularidade**, as funcionalidades de um sistema devem ser divididas em vários serviços; e (vi) **Distribuição**, os serviços devem ficar distribuídos, para aproveitar melhor os recursos computacionais [13].

Considerando que um serviço pode ser uma interface de um agente, podendo ser solicitado por um software ou agente, resultando na troca de mensagens, executando as atividades e interagindo via protocolo de troca de mensagens definida na descrição do serviço, pode-se então visualizar agentes em um conceito de SOA. Assim, são definidas entidades que atuam para a execução de determinada atividade ou para atingir algum objetivo. Por mais que se tratem de metodologias específicas, ambas trabalham com a noção de atividade, objetivo, tarefa e interação orientada a mensagens [11].

Para que os agentes possam trabalhar com SOA, deve haver três características: (i) um agente deve poder descobrir serviços e se adaptar a eles; (ii) devem existir serviços web com a descrição do protocolo para que um agente possa invocá-lo e; (iii) um agente deve ser capaz de realizar interações complexas com vários serviços ao mesmo tempo [7].

Assim, desenvolvedores podem encontrar e utilizar agentes pela invocação de serviços web. Tal integração traz benefícios imediatos em que torna um serviço web capaz de invocar um agente de serviço e vice-versa, permitindo assim, através dos conceitos e tecnologias de agentes, novas e avançadas funcionalidades na utilização de SOA [4].

III. GERENCIADOR DE COMPORTAMENTOS

Basicamente, a forma como as informações estão estruturadas/relacionadas são a base para a execução do algoritmo do agente. Dessa forma, tal estrutura é a primeira parte apresentada na proposta. A forma como o agente é executado está diretamente ligada com essa estrutura, permitindo que o usuário possa personalizar a ordem e forma da execução das operações em tempo de execução. Por fim, é mostrado brevemente o editor que permite que essa estrutura possa ser configurada para então ser executada pelo software servidor.

A. Estrutura das Informações

A Estrutura das Informações, visualizadas em um modelo entidade-relacionamento (ER) na Fig. 1., é utilizada para organizar as informações e a execução dos comportamentos.

A entidade *Service* é armazena os serviços web que podem compor os comportamentos do agente. Por meio desta, o sistema busca o WSDL do serviço web e apresenta para o usuário suas operações.

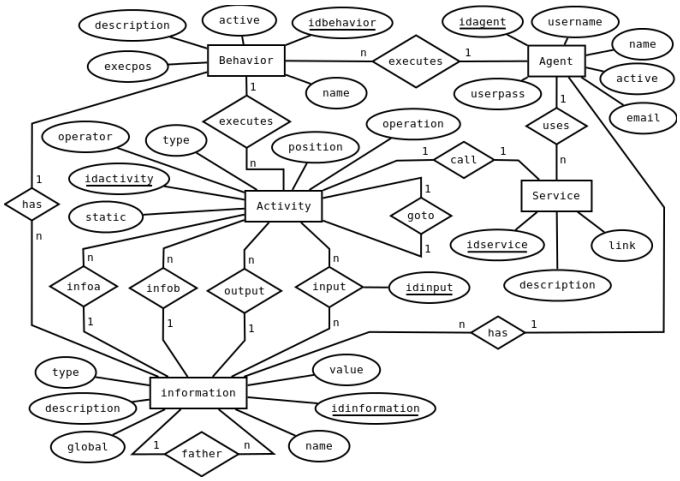

Fig. 1. Diagrama Entidade-Relacionamento do Sistema de Gerenciamento de comportamento dos agentes.

No protótipo, um agente possui uma estrutura de informações na forma de árvore, armazenadas na entidade *Information*. Uma informação pode ter várias informações filhas e cada filha pode ter outras diversas filhas. Para manter essa relação, em um atributo chamado *father* é definida a informação pai. Para a informação raiz, *father* é *null.* As informações podem ser de diferentes tipos, definidos no atributo *type*. Os tipos podem ser básicos como *string*, *integer*, *float*, *boolean*, *text*, ou um tipo complexo chamado *struct*. Se o tipo for *struct*, então a informação é caracterizada como pai, podendo conter *n* informações filhas. Para que uma informação possa ser visualizada e utilizada por outros comportamentos do agente, atribui-se *true* para o atributo *global* da informação. Caso contrário esta só pode ser visualizada no comportamento da qual a informação pertence, identificado em *idbehavior*.

Um agente pode possuir diversos comportamentos executando concorrentemente. Para que um comportamento entre em execução, deve ser atribuído *true* ao atributo *active* em *Behavior*. A estrutura de execução dos comportamentos proposta neste trabalho segue um formato de algoritmo. Cada linha de execução é uma atividade e está colocada em sua devida ordem. O atributo *execpos* se referencia à atividade em que a execução do comportamento se encontra. Se o usuário requisitar a interrupção de um comportamento, a posição de execução é salva para que o este possa voltar a se executar a partir da posição que parou.

As linhas de execução dos comportamentos são chamadas de atividades (*Activity*). São elas:

- ***assign call***: invoca um serviço web. Em *service* há um link para o serviço e em *operation* a operação. O *output* é ligado à uma *Information* e recebe o retorno da invocação do serviço. Os atributos *inputs* (*Information*) são os parâmetros passados à operação do serviço;
- ***assign call var***: funciona como uma *assign call*, mas utiliza o link para o serviço web e a operação na forma de variáveis (*Information*);
- ***assign static***: uma variável recebe um valor passado de forma estática e armazenado no atributo *static* da entidade *Activity*;
- ***assign var***: uma variável (*infoa*) recebe o valor de outra variável (*infob*);
- ***conditional static***: se uma condição (se-então) é satisfeita, o bloco interno é executado. Os atributos utilizados para efetuar a validação são *infoa*, *operator* (==, !=, <, >, <=, >=, contem) e *static* (valor estático);
- ***conditional var***: funciona tal como a atividade anterior, mas a segunda informação também é uma variável;
- ***conditional end***: aponta (*goto*) para o local onde o bloco condicional inicia. As atividades *conditional static* e *conditional var* precisam possuir uma ligação (*goto*) ao seu *conditional end*.
- ***loop static***: funciona como o *conditional static*, mas o bloco continua sendo executado enquanto a condição é satisfeita.
- ***loop var***: funciona como o *conditional var,* mas continua executando enquanto a condição é satisfeita.
- ***loop end***: Indica fim de bloco de um laço e deve ser ligado (*goto*) com seu início de bloco de laço também.

Essas informações são utilizadas pelo servidor para executar os comportamentos dos agentes, O usuário pode criar, alterar e excluir comportamentos por meio de uma interface web de configuração.

B. *Software Servidor*

O protótipo baseado na proposta desse artigo foi desenvolvido para suportar múltiplos usuários, cada qual com seu próprio agente. O servidor se mantém em um laço de execução, avaliando a cada período de tempo se o agente está ativo. Se um agente estava em execução e seu atributo *active* é alterado para *false*, significa que o agente foi desligado. O servidor envia uma mensagem para o agente avisando que o mesmo foi desligado. O agente salva suas informações e termina sua execução.

Cada agente é uma *thread* que possui uma lista de comportamentos e sua execução se dá na forma de um laço que efetua o gerenciamento desses comportamentos.

A utilização de *threads* se dá pois essas possuem suporte há múltiplas linhas de controle dentro de um processos. Elas ocupam o mesmo espaço de endereçamento, mas cada uma possui seu próprio ponteiro de controle de execução [14]. Um programa não precisa ficar esperando o retorno para continuar trabalhando em outras tarefas. Ou seja, podem haver vários agentes sendo executados simultaneamente no servidor. Os comportamentos (*Behavior*) dos agentes são compostos de várias linhas de execução (tal como um algoritmo) e precisam ser executados em paralelo. Nesse caso, também foram implementados como *threads*.

Os comportamentos se mantém em execução, executando cada uma de suas atividades. Há um ponteiro de execução (*execpos*) que indica qual atividade está em execução. Quando esta é executada, o ponteiro pula para a próxima atividade. O que define qual atividade é executada na sequência depende do retorno da execução da atividade corrente. Por exemplo, se uma atividade simples de atribuição estática *assign static* é chamada, ela retorna a posição dela acrescida de um, ou seja, a atividade seguinte. Se for um tipo condicional, a posição retornada é, ou a primeira atividade interna do bloco (se válida a condição), ou a próxima posição após seu final de bloco (se inválida). No caso do laço, retorna ou a primeira posição do bloco, ou a próxima posição do final de bloco e no final do bloco do laço, a posição é direcionada para o início do bloco.

Quando *execpos* possui um valor maior do que a quantidade de atividades que o comportamento possui, então o valor de *execpos* retorna para 1, e a execução do algoritmo recomeça.

C. *Interface Web de Configuração*

A interface web de gerenciamento do agente serve para criar um agente, adicionar informações, serviços e comportamentos. Cada comportamento pode ser ativar ou desativar.

Nessa interface o usuário pode cadastrar um novo serviço web para ser utilizado nos comportamentos do agente, alterar os existentes, ou visualizá-los. O usuário pode visualizar detalhes desses serviços, com suas operações e um link para o documento WSDL do serviço web.

A edição dos comportamentos (Fig. 2.) possui (à esquerda) informações específicas do comportamento (nome e descrição). Também há dois botões, o botão de ligar/desligar comportamento e o botão de recarregar informações na página.

Ao clicar no link “Detalhes”, aparece a lista de informações do comportamento e a lista das informações globais. O usuário pode acrescentar novas informações, editar ou excluir. Observa-se que informações que estão sendo referenciadas por alguma atividade não podem ser excluídas. Localizado na parte direita da Fig. 2. está o comportamento na forma de algoritmo.

Fig. 2. Editor de Comportamentos do agente.

Para adicionar uma nova atividade em um comportamento, o usuário deve clicar no ícone “+” nas linhas de atividades. O

ícone "-" serve para excluir uma atividade. Caso uma atividade seja um bloco, apenas o início do bloco e o final são excluídos, restando os elementos internos.

As setas servem para alterar a ordem de uma atividade. Ao clicar a seta para cima, uma atividade é trocada pela atividade anterior, se clicado na seta para baixo, a atividade é trocada pela posterior. Quando um início ou fim de bloco é movido, todo o bloco segue junto. Se ao mover um bloco para cima, for encontrado o final de outro bloco, então o bloco que foi movido é inserido dentro do bloco superior.

Quando o usuário passa com o mouse por cima de informações e outros itens da atividade, outras informações são apresentadas, como o valor de uma informação, o link de um serviço web, etc.

A atividade que se encontra em execução, no momento em que a página é carregada, é apresentada com uma cor azul claro. Para que o usuário possa visualizar o andamento da execução, é necessário clicar no botão recarregar, na parte esquerda da página. Neste protótipo, as informações não são carregadas automaticamente.

O ícone de edição, entre o "-" e a seta para cima, serve para editar um comportamento existente. Para cada tipo de atividade diferente, um formulário diferente é apresentado. No caso, é a edição de uma atividade de chamada de serviço web (Linha 2).

Um campo do formulário é reservado para a informação que deve receber o retorno da invocação do método, o segundo campo é o link do serviço e o terceiro a operação. Ao colocar ou alterar essas informações, deve ser selecionado o botão "ok" para salvar os dados, antes mesmo de selecionar os parâmetros. Em "Parâmetros:" há uma lista de parâmetros que já foram inseridos, na ordem que deve estar na operação do serviço web. Também à um botão de excluir, para retirar um parâmetro já inserido. Para adicionar um novo parâmetro, basta selecionar selecionar da lista de informações do comportamento e informações globais e clicar no botão "adicionar".

Quando uma atividade do tipo condicional ou laço é criada, automaticamente, e logo na sequencia, uma atividade de fechamento de bloco é criado (FIM SE ou FIM ENQUANTO).

IV. Considerações Finais

Este artigo apresentou uma proposta para edição do comportamento de agentes que permite que o usuário possa personalizar o agente para suas necessidades, em tempo de execução e sem que precise uma recompilação do agente.

Ainda, a proposta permite que o usuário possa configurar seu agente para efetuar chamadas a serviços web. Isso permite ao agente executar operações remotas, distribuindo e paralelizando o processamento e dando maior poder computacional. Essas operações também podem ser implementadas e fornecidas por terceiros. Além disso, por meio das chamadas a serviços web, o agente pode efetuar processos de negócios para os usuários dos agentes. Em tempo, a estrutura de algoritmos da proposta permite que possa ser efetuada uma orquestração de serviços web no agente.

Para avaliar a proposta em funcionamento, um protótipo foi desenvolvido. Este foi dividido em duas partes: (i) um editor de comportamentos que pode se utilizar de chamadas de serviços web para compor as atividades dos algoritmos dos comportamentos dos agentes, tal como chamadas de funções em linhas de execução em linguagem de programação e; (ii) um software servidor que mantem os agentes em execução, rodando paralelamente cada agente e cada comportamento desses agentes.

Os próximos passos podem seguir em diversos vieses. Não obstante, este artigo limita-se aqui a uma sugestão inicial, tal como a criação de uma interface de configuração dos comportamentos de forma gráfica, ou seja, a utilização de elementos de fluxogramas para compor os comportamentos ao invés de uma estrutura na forma de algoritmos. Tal recurso visa a facilitar a utilização do editor por usuários que não entendem de programação, mas que conseguem entender a lógica de um fluxograma de execução.

Referências

[1] Booth, D.; Haas, H.; McCabe, F. (2004). Newcomer, E.; et al. "Web Services Architecture". Disponível em: <http://www.w3.org/TR/ws-arch/> Acessado em Março/2013.

[2] Bush, J.; Irvine, J.; Dunlop, J. (2006). "Personal Assistant Agent and Content Manager for Ubiquitous Services". Wireless Communication Systems, 2006. ISWCS'06. 3rd International Symposium on. pg.169-173.

[3] Estefan, J. A.; Laskey, K.; McCabe, F.; Thorthon, D. (2008). "Reference Architecture for Services Oriented Architecture Version 1.0". OASIS. Disponível em: <http://docs.oasis-open.org/soa-rm/soa-ra/v1.0/soa-ra-pr-01.pdf>. Acessado em: Março/2013.

[4] Greenwood, D.; Calisti, M.; at al. (2004). "Engineering Web Service - Agent Integration". IEEE International Conference on Systems, Man and Cybernetics. v2.

[5] Haas, H e Brown, A. (2004). "Web Services Glossary". Disponível em: <http://www.w3.org/TR/ws-gloss/>, Acessado em Março/2013.

[6] Huang, Y. e Chung, J.Y. (2003). "A web services-based framework for business integration solutions". Electronic Commerce Research and Applications, 2(1):15–26. Disponível em: <http://www.sciencedirect.com/science/article/B6X4K-48642HS-1/2/a751ffc1be1676f5b9955ea9050c160d>, acessado em Fevereiro/2013.

[7] Kuno, H. and Sahai, A.. (2002). "My agent wants to talk to your service: personalizing web services through agents". Proceedings of the First International Workshop on Challenges in Open Agent Systems. Pg 25-31.

[8] MacKenzie, C.; Laskey, K.; McCabe, F. at all. (2006). "Reference Model for Service Oriented Architecture 1.0". OASIS Standard. Disponível em: <http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf>. Acessado em Fev/2013.

[9] Michaell, T., Caruana, R., Freitag, D., McDermott, J., Zabowski, D. (1994). "Experience with a learning personal assistant". Communications of the ACM, July.

[10] O'Brien, L.; Bass, L.; Merson, P. (2005). "Quality attributes and service-oriented architectures". Technical Note - Software Engineering Institute, Carnegie Mellon University.

[11] Ricci, A.; Buda, C.; Zaghini, N.. (2007). "An agent-oriented programming model for SOA & web services". Industrial Informatics, 5th IEEE International Conference on. v2.

[12] Sensoy, M. and Yolum, P.. (2008). "Evolving service semantics cooperatively: a consumer-driven approach". Springer Science+Business Media, LLC, Nov 9.

[13] Singh, M.P. and Huhns, M.N.. (2005). "Service-oriented computing: semantics, processes, agents". John Wiley & Sons, New York, NY, EUA.

[14] Tanenbaum, A.S.. (2010). "Sistemas Operacionais Modernos". Prentice Hall - Br. 3ed.

[15] Weiss, G.. (1999). "Multiagent systems: a modern approach to distributed artificial intelligence". MIT Press.