AgeODE: Uma Infra-estrutura para Apoiar a Construção de Agentes para Atuarem no Ambiente de Desenvolvimento de Software ODE

Juliana Pezzin, Ricardo de Almeida Falbo

Mestrado em Informática, Universidade Federal do Espírito Santo, Vitória - ES - Brasil
juliana_pezzin@hotmail.com, falbo@inf.ufes.br

Abstract. Since software agents provide solutions to complex computational problems, Software Development Environments (SDEs) can benefit themselves from this technology. This paper presents an infrastructure (AgeODE) to support the development of agents to run into a SDE called ODE, Ontology-based software Development Environment. The infrastructure defines four main classes of agents potentially useful in the context of SDEs, the way communication between agents occurs, how agents access SDE´s objects that are part of their knowledge bases, and how the agents' internal architecture is. It also presents a case study using the proposed infrastructure.

1 Introdução

Desenvolver software de qualidade assegurada, com elevada produtividade, dentro dos prazos e custos estabelecidos e sem necessitar de mais recursos do que os alocados, tem sido um grande desafio para a Engenharia de Software. Neste contexto, a capacidade de agentes de software para, de forma autônoma, executar (ou apoiar a realização de) algumas tarefas tem estimulado bastante o uso dessa tecnologia na automatização do processo de software.

Uma vez que Ambientes de Desenvolvimento de Software (ADSs) são sistemas computacionais de apoio ao desenvolvimento e à manutenção de produtos de software, e à gerência das atividades envolvidas neste contexto, tais ambientes são potenciais beneficiários da tecnologia de agentes. Dentre as características gerais de agentes que os tornam atrativos no contexto de ADSs, podem ser citadas: (i) a habilidade para agir com autonomia; (ii) representação de auto-nível do comportamento – um nível de abstração acima das construções orientadas a objetos; (iii) flexibilidade, combinando características comportamentais pró-ativas e reativas; (iv) desempenho em tempo real; (v) adequação para aplicações distribuídas; e (vi) habilidade para trabalhar cooperativamente.

Entretanto, deve-se ressaltar que a integração é uma questão fundamental para ADSs [1, 2] e, portanto, a construção de agentes para atuarem em um ADS deve considerar que os agentes desenvolvidos devem apresentar homogeneidade na forma de apresentação, comunicação e atuação. Tendo isso em mente, foi desenvolvida

AgeODE, uma infra-estrutura para apoiar a construção de agentes para atuarem no ambiente ODE (***O****ntology-based software* ***D****evelopment* ***E****nvironment*) [2].

Ainda que existam várias infra-estruturas de apoio à construção de agentes, de modo geral, elas não se destinam a ser especificamente uma infra-estrutura para construção de agentes para um ADS e, conseqüentemente, não tratam aspectos relacionados à integração de agentes em um ADS, tal como uniformidade. Assim, buscando preencher essa lacuna, foi construída a infra-estrutura apresentada neste trabalho.

Este artigo está estruturado da seguinte forma: a seção 2 introduz o conceito de agentes e discute como requisitos de ADSs podem ser melhor tratados com o uso da tecnologia de agentes. Essa seção aponta, ainda, algumas infra-estruturas para construção de agentes e ADSs que fazem uso dessa tecnologia. São discutidas, também, as primeiras iniciativas de uso de agentes em ODE e os problemas encontrados. A seção 3 apresenta AgeODE, a infra-estrutura para construção de agentes proposta e discute como ela trata os problemas apontados. A seção 4 mostra, por meio de um estudo de caso, como a infra-estrutura proposta é utilizada na construção de agentes em ODE. Finalmente, na seção 5, são apresentadas as conclusões deste trabalho.

2 Agentes e Ambientes de Desenvolvimento de Software

Para um grande número de aplicações, são necessários sistemas capazes de decidir por eles mesmos o que precisa ser feito. Tais sistemas são conhecidos como *agentes*, que podem ser vistos como sistemas capazes de ações autônomas em algum ambiente, a fim de atingirem seus objetivos de projeto. Um agente tipicamente percebe seu ambiente através de sensores e tem disponível um repertório de ações que podem ser executadas para modificar o ambiente, o qual pode responder de forma não-determinística à execução de suas ações [3].

A utilização de agentes para a solução de problemas complexos de natureza distribuída e descentralizada tem sido bastante explorada por trabalhos de pesquisa realizados nas últimas décadas. Muito mais que uma nova tecnologia, a orientação a agentes (OA) tem se mostrado uma nova forma de pensar a estratégia de solução de problemas. Isso tem acontecido porque a abordagem de agentes, por suas características de distribuição e descentralização do controle, permite desenvolver soluções para problemas complexos de maneira mais natural que soluções monolíticas estruturadas ou orientadas a objetos (OO), tradicionalmente adotadas.

Uma vez que o uso da tecnologia de agentes é indicado no desenvolvimento de sistemas complexos, Ambientes de Desenvolvimento de Software (ADSs) são potenciais candidatos a explorar os benefícios dessa tecnologia.

Tendo em vista a complexidade e diversidade de problemas enfrentados na construção de um ADS, pode-se refletir e questionar: Por que não usar agentes em um ADS? O que poderia ser mais facilmente realizado em um ADS com o apoio de agentes? Analisando requisitos de ADSs propostos em [1], é possível apontar alguns que poderiam ser mais facilmente satisfeitos através do uso de agentes:

- *Ser customizável, de uso amigável e oferecer suporte a diferentes tipos de usuários:* agentes podem ser usados para perceber o ambiente e detectar preferências e características dos usuários. Com base nesse perfil, o ADS pode ser customizado e sua interface adaptada às características de um usuário específico.
- *Oferecer suporte ao trabalho cooperativo:* agentes podem cooperar para realizar certas atividades. Eles são capazes de interagir com outros agentes (e possivelmente humanos), buscando atingir seus objetivos. Os agentes podem ser uma ponte entre o ADS e os usuários, facilitando a cooperação e coordenação do trabalho a ser realizado, através do envio de avisos e mensagens do ADS para o usuário, apoiando a comunicação entre os usuários.
- *Oferecer suporte à gerência do projeto:* agentes, de forma autônoma e pró-ativa, podem monitorar o progresso e capturar as características e o estado do projeto, de modo a informar ao gerente de projeto sobre o andamento do projeto, as metas que foram alcançadas etc, sem que este se preocupe em buscar as informações do projeto em questão, deixando-o mais livre para a tomada de decisões.
- *Oferecer suporte à gerência de conhecimento:* agentes podem ser utilizados para ligar os membros da organização ao conhecimento disponível. Eles podem auxiliar não apenas na busca por conhecimento, mas também no filtro de conhecimento relevante e sua disseminação [4].
- *Oferecer assistência inteligente ao usuário:* agentes podem funcionar como tutores, aconselhando ou avisando o usuário das tarefas que ele pode realizar e como realizá-las.
- *Oferecer suporte para medição e avaliação do produto:* agentes podem apoiar a coleta automática de métricas e compará-las com informações de projetos passados de modo a facilitar o acompanhamento de projeto.

Tendo em vista que agentes podem ser bastante úteis neste contexto, é importante investigar maneiras de se introduzir essa tecnologia em um ADS. Deve-se ressaltar que a integração de agentes em um ADS deve ser feita respeitando as várias dimensões de integração [1, 2], o que inclui formas uniformes de atuação, apresentação e comunicação.

Há várias infra-estruturas de agentes provendo ferramentas para construção de agentes descritas na literatura, dentre elas JATLite [5] e JADE [6]. Entretanto, nenhuma delas se destina a ser uma infra-estrutura específica para construção de agentes em um ADS, desconsiderando, portanto, aspectos importantes de integração.

Por outro lado, alguns ADSs fazem uso da tecnologia de agentes, tais como ODE [2], PROSOFT [7] e Odyssey [8]. Entretanto, nesses casos, não havia uma infra-estrutura definida para apoiar a construção e integração de agentes no ADS.

Seja o caso do ambiente ODE. ODE (***O****ntology-based software* ***D****evelopment* ***E****nvironment*) [2, 9] é um Ambiente de Desenvolvimento Centrado em Processo, desenvolvido no Laboratório de Engenharia de Software da Universidade Federal do Espírito Santo (LabES/UFES), tendo por base ontologias. A integração é tratada como uma questão fundamental nesse ambiente e, para facilitá-la, o ambiente e suas ferramentas compartilham algumas ontologias de engenharia de software, tal como a ontologia de processo de software [10]. Uma vez que as ferramentas do ambiente são construídas baseadas em ontologias, a integração dessas ferramentas é facilitada, pois os conceitos envolvidos estão bem definidos. Assim, a mesma ontologia é usada para construir diferentes ferramentas que suportam atividades relacionadas da engenharia

de software, reduzindo confusões terminológicas e facilitando o entendimento compartilhado e a comunicação entre as ferramentas que compõem o ambiente [2].

Dentre as ontologias adotadas em ODE, uma merece destaque: a ontologia de processo de software [10], usada como base para a infra-estrutura de controle de processos de software de ODE [11]. Essa infra-estrutura permite definir processos de software e controlar seus respectivos projetos e, integradas a ela, existem diversas ferramentas de apoio à construção, gerência e avaliação de qualidade, que auxiliam a realização de diversas atividades do desenvolvimento de software.

Para facilitar a integração, as ferramentas desenvolvidas no contexto de ODE utilizam, além da infra-estrutura de controle de processos de software, uma série de facilidades oferecidas pelo ambiente, dentre elas: a infra-estrutura de conhecimento [9], que disponibiliza classes representando os elementos mais comuns ao domínio de engenharia de software, construídas com base nas ontologias; a infra-estrutura de gerência de conhecimento [4], que disponibiliza funcionalidades de gerência de conhecimento; a camada de persistência, que gerencia o armazenamento de dados no repositório do ambiente; a camada de inferência [11], que disponibiliza serviços de inferência para o ambiente; além de diversos utilitários de interfaces gráficas.

As primeiras iniciativas de se utilizar agentes em ODE foram feitas no sentido de apoiar a customização do ambiente para usuários específicos e a realização de sub-atividades do planejamento, tal como a alocação de recursos. Posteriormente, agentes foram usados para apoiar a disseminação de conhecimento na infra-estrutura de gerência de conhecimento de ODE [4]. No desenvolvimento desses agentes, alguns problemas foram detectados, a saber: (i) agentes não possuíam autonomia, uma vez que eram tratados como objetos e, portanto, não possuíam linha de controle própria nem eram tratados como processos separados; (ii) cada agente era construído de uma maneira diferente por cada um dos desenvolvedores e, deste modo, não havia padronização nem uniformidade na construção dos agentes, o que provocava problemas de integração; (iii) finalmente, cada desenvolvedor partia do zero na árdua tarefa de se construir agentes, não havendo, deste modo, nenhuma forma de reuso.

Visando tratar esses problemas, foi desenvolvida AgeODE, a infra-estrutura apresentada neste trabalho. Tal infra-estrutura permite que agentes sejam desenvolvidos de uma forma padronizada e mais ágil, fazendo com que eles compartilhem diversas funcionalidades, tal como a forma de comunicação. A aplicabilidade da tecnologia de agentes para o desenvolvimento de funcionalidades de um ADS, a inexistência de uma infra-estrutura voltada especificamente para este fim e a grande preocupação com a questão da integração em ODE serviram de motivação para o desenvolvimento de AgeODE.

3 AgeODE: Uma Infra-estrutura para Apoiar a Construção de Agentes para o Ambiente ODE

No desenvolvimento de um sistema, agentes e objetos podem ser usados para modelar diferentes entidades do domínio do problema. Agentes são concebidos para representar entidades ativas que manipulam objetos, que, por sua vez, são comumente entidades passivas no sistema de software. Além disso, agentes podem apresentar

características bastante distintas, o que conduz a uma classificação de agentes, onde cada tipo de agente tipicamente inclui diferentes interesses.

A infra-estrutura AgeODE considera esses aspectos. Assim, agentes têm acesso aos objetos existentes em ODE, que podem ser vistos como parte de suas bases de conhecimento, e cooperam e negociam com outros agentes para alcançarem seus objetivos. Além disso, buscando uniformidade, uma arquitetura básica para os agentes de ODE é definida e alguns tipos de agentes importantes para ADSs são providos.

Uma vez que construir uma infra-estrutura desta natureza é uma tarefa complexa, visando ao reuso, AgeODE foi definida como uma camada sobre a infra-estrutura JAT*Lite* [5], utilizando algumas de suas classes, principalmente para tratar a comunicação entre agentes.

A seguir, discute-se como AgeODE se apóia em JAT*Lite* e são apresentados os tipos de agentes que compõem AgeODE, como se dá a comunicação entre agentes, como os agentes acessam os objetos de ODE e como a infra-estrutura foi implementada.

3.1 AgeODE e JAT*Lite*

Conforme anteriormente citado, AgeODE foi definida como uma camada sobre JAT*Lite* [5], visando reutilizar, sobretudo, os elementos necessários para tratar a comunicação entre agentes.

A comunicação entre agentes em JAT*Lite* se baseia em um modelo cliente-servidor: agentes clientes usam o serviço de roteamento oferecido por um agente servidor, denominado roteador. Assim, ligadas às classes de JAT*Lite*, há duas classes de AgeODE: *AgCliente* e *AgRoteador*, como mostra a figura 1.

AgCliente, ao herdar da classe *RouterClientAction* de JAT*Lite*, incorpora a AgeODE as características de agentes clientes de JAT*Lite*, oferecendo serviços para enviar e receber mensagens de outros agentes. *AgRoteador*, descendente da classe *RouterAction* de JAT*Lite*, funciona como um roteador de mensagens e fica aguardando agentes clientes se conectarem a ele. Esse tipo de agente fornece serviços para nomear, endereçar e localizar agentes em um sistema multiagente. Com um roteador, agentes não têm que saber o endereço de um outro agente nem como se comunicar com ele. Essas tarefas ficam a cargo do roteador, que funciona como uma ponte de comunicação entre os agentes ligados a ele [12]. Em uma aplicação multiagente, pode-se ter um ou vários roteadores, sendo que cada agente no sistema deve especificar a qual agente roteador está associado.

3.2 Tipos de Agentes Clientes de AgeODE

Por ser uma infra-estrutura genérica para a construção de agentes, JAT*Lite* não define outras classes de agentes; somente os separa em clientes e roteadores. Entretanto, no contexto de ADSs, é interessante prover um conjunto básico de classes de agentes abrangendo as formas de atuação mais comuns nesse contexto. Assim, outros tipos de agentes foram especializados em AgeODE.

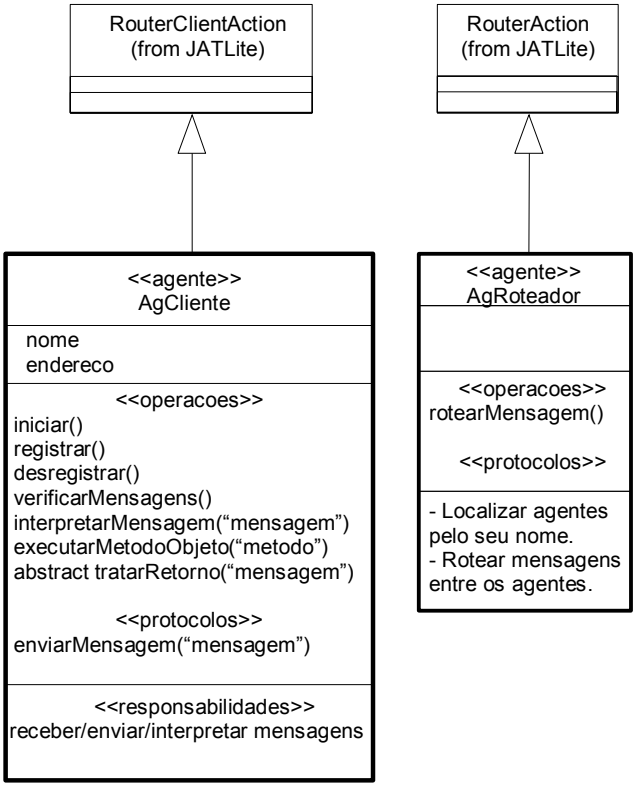

Fig. 1. Tipos básicos de agentes de AgeODE e sua relação com as classes de JAT*Lite*.

A partir da análise de alguns padrões de agentes, classificações de agentes e trabalhos relacionados [12, 13, 14, 15], foram propostas quatro classes de agentes clientes para AgeODE, mostradas na figura 2. A seguir, são descritos esses tipos de agentes, sendo que vale ressaltar que um agente específico de um sistema pode assumir os perfis de mais de um tipo de agente cliente.

- **Agente de Interface:** Um agente de interface visa fornecer ao usuário do ADS uma interface mais amigável, com características pró-ativas, detectando as ações do usuário quando usando a interface do ambiente. Ao gerenciar interfaces gráficas com o usuário, um agente desse tipo transforma comandos do usuário em objetivos, realiza ações e exibe os resultados [14]. Assim, ao observar um usuário e suas ações, pode adaptar a interface às necessidades identificadas pelo perfil do usuário, estabelecendo padrões de uso mais adequados.

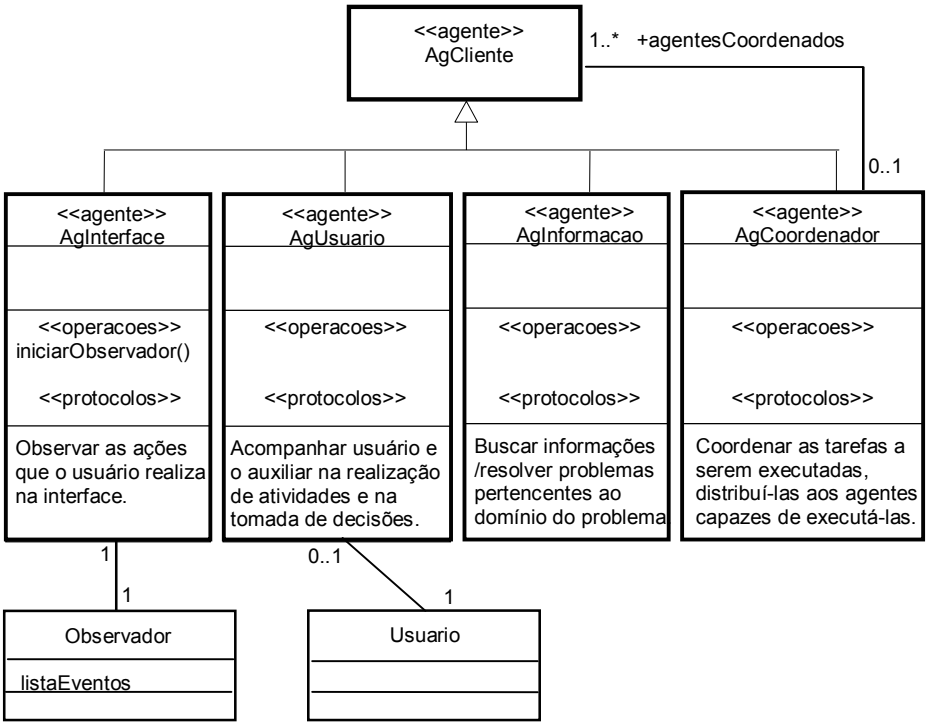

Fig. 2. Tipos de agentes clientes de AgeODE.

- **Agente de Usuário:** Uma vez que um ADS é usado por vários usuários, é importante que o ambiente possua uma forma de distingui-los de acordo com suas características, ou seja, de acordo com os papéis que desempenham, tais como gerente de projeto, engenheiro de software, gerente de configuração etc, e seus interesses de uso do ambiente. Um agente de usuário deve usar o conhecimento que possui sobre determinado usuário para permitir ou não a execução de atividades por ele. Ele deve ser capaz de estabelecer o perfil do usuário, buscando informações pertinentes ao usuário no contexto do ADS ou de um projeto específico. Um agente de usuário pode interagir com o usuário que ele representa, auxiliando-o na realização de atividades e na tomada de decisões. É como se esse agente incorporasse o papel do usuário, passando a realizar algumas de suas atividades e ajudando-o na realização de outras.
- **Agente de Informação (Agente de Domínio):** Agentes de informação são responsáveis pela realização de alguma funcionalidade do sistema. Sua tarefa principal é buscar informações e realizar tarefas dentro do domínio de problemas do ADS. Esse tipo de agente disponibiliza uma interface contendo um conjunto de serviços que podem ser requisitados por outros agentes do ADS ou para seu uso próprio.

- **Agente Coordenador:** Este agente tem como objetivo coordenar as tarefas a serem executadas em um dado momento pelos agentes do ADS. Para tal, dentre outros, deve ser capaz de: distribuir tarefas aos agentes, consolidar resultados de tarefas e/ou conjuntos de informações recuperadas de um ou mais agentes dispersos na sociedade e conhecer os agentes (e suas capacidades/habilidades específicas) que estão sob seu domínio de coordenação.

Visando uniformidade e, por conseguinte a integração, os agentes clientes de AgeODE têm de compartilhar uma arquitetura básica. Assim, os agentes clientes construídos usando AgeODE possuem as seguintes características:

- Atributos gerais, obrigatórios para todos os agentes clientes, o que inclui: nome e endereço;
- Outros atributos, que podem ser objetos de ODE de seu interesse, uma vez que esses constituem parte de sua base de conhecimento;
- Operações, representando atividades dos agentes, isto é, computações feitas pelo agente sem a necessidade de interagir com outros agentes;
- Protocolos, que definem como um agente pode interagir com outros agentes. Toda comunicação entre agentes clientes requer a execução do protocolo básico `enviarMensagem`;
- Autonomia, fazendo com que cada agente possua sua própria linha de controle. A autonomia básica dos agentes é dada por JAT*Lite*. Outras questões relacionadas à autonomia dos agentes são tratadas em AgeODE, tal como a capacidade de observar o ambiente;

Os agentes clientes, por herdarem de *AgCliente*, podem, ainda: iniciar sua atuação no ambiente (executando a operação `iniciar`); se registrar ou cancelar seu registro em um roteador (executando, respectivamente, as operações `registrar` e `desregistrar`); verificar se chegaram novas mensagens (executando de tempos em tempos a operação `verificarMensagens`) e, havendo mensagens, interpretar cada uma delas (executando a operação `interpretarMensagem`); e finalmente, agir de acordo com o conteúdo da mensagem (executando a operação `tratarRetorno`).

É importante, ainda, comentar que os agentes do tipo interface (*AgInterface*), por necessitarem capturar eventos da interface com o usuário, têm a eles associados um objeto *Observador*. Esse objeto *Observador* age como se fosse o mecanismo de percepção de um agente de interface no ambiente ODE, capturando os eventos das interfaces com o usuário do ambiente. Através de seus observadores, os agentes de interface sabem o que os usuários estão fazendo em ODE, sendo, portanto, a forma desses agentes estarem observando o ambiente. Um objeto observador é único para um agente, pois cada agente possui funcionalidades próprias e possui necessidades diferentes de observação do ambiente. Esse observador é o responsável por capturar os eventos pré-determinados da interface que sejam relevantes para o agente e, assim, os eventos que um agente de interface precisa perceber têm de estar definidos no seu correspondente observador. Quando um evento desses ocorre, o observador avisa ao agente, para que esse possa ficar a par do que está acontecendo no ambiente e, assim, possa tomar suas ações.

3.3 A Comunicação entre Agentes

Para contemplar a integração, o projeto dos agentes de um ADS deve levar em consideração a necessidade de se estabelecer um padrão para a comunicação. A linguagem adotada para a comunicação entre os agentes em AgeODE foi KQML *(Knowledge Query and Manipulation Languag*e) [16], uma vez que JAT*Lite* [5] já adota essa linguagem. KQML é uma linguagem formal, com um conjunto de protocolos que apóiam a comunicação e promovem a troca de informações e conhecimento, suportando uma comunicação de alto nível entre agentes [16].

As primitivas da linguagem são chamadas de *performativas* ou *atos de fala*, que definem as ações admissíveis que agentes podem tentar na comunicação com outros agentes. A sintaxe de KQML é baseada numa lista de argumentos. O elemento inicial da lista é a performativa. Os elementos restantes são os parâmetros da performativa, representados por pares palavra-chave/valor. As performativas implementadas em AgeODE são: *ask-one*, *broadcast*, *register*, *unregister*, *reply*, *sorry* e *tell* [16].

Uma vez que, no que concerne à comunicação, AgeODE utiliza a infra-estrutura JAT*Lite*, as mensagens KQML em AgeODE, como em JAT*Lite*, são implementadas da seguinte forma: cada mensagem é uma estrutura que possui vários campos do tipo *strin*g, um para cada parâmetro da mensagem. Para compor uma mensagem KQML, um agente deve preencher os campos correspondentes da mensagem e enviá-la. Ao receber uma mensagem de outro agente, o agente receptor a interpreta conforme as diretrizes de KQML.

Como ODE é um ADS baseado em ontologias, para facilitar e padronizar a comunicação entre agentes, as mensagens têm de descrever o parâmetro *:ontology*, informando as ontologias envolvidas na comunicação. Assim, para que um agente consiga se comunicar com outro, eles devem conhecer a(s) mesma(s) ontologia(s).

Conforme discutido anteriormente e mostrado na figura 3, agentes, ao se comunicarem entre si, não enviam mensagens diretamente uns para os outros. Ao contrário, há um agente do tipo roteador que trata as mensagens encaminhadas pelos agentes que estão sob seu domínio.

3.4 A Comunicação entre Agentes e Objetos

Muitas vezes, objetos do ambiente compõem parte da base de conhecimento dos agentes. Assim, é necessário prover um mecanismo padrão para acesso a objetos por parte dos agentes. Esse mecanismo é ilustrado também na figura 3.

A comunicação entre agentes e objetos não exige que haja um protocolo de comunicação, como ocorre na comunicação entre agentes. No caso da comunicação entre agentes e objetos, agentes invocam métodos dos objetos diretamente, abrindo um canal de comunicação entre um agente e ODE, usando *sockets*. Um *socket* pode ser visto como um *"bocal virtual"*, onde agentes e objetos podem se conectar para introduzir e retirar bits, sendo que existe um *socket* nos dois pontos finais de uma conexão. Vale destacar que esse procedimento é transparente para um desenvolvedor de agentes usando AgeODE, pois ele só precisa utilizar os métodos que obtêm objetos de ODE.

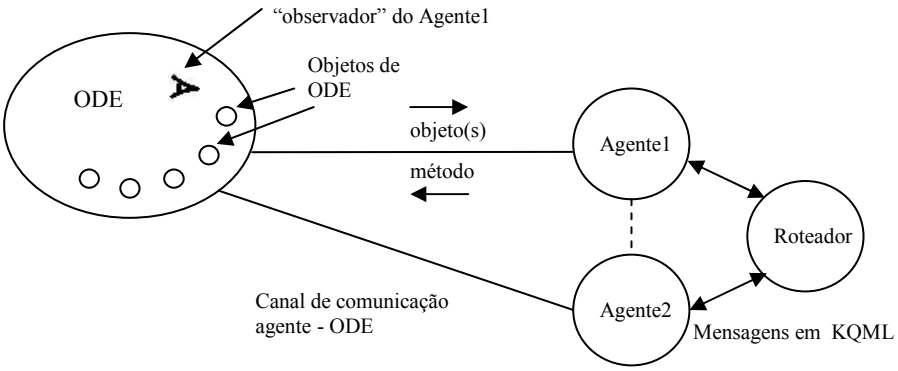

Fig. 3. A comunicação entre agentes e objetos de ODE.

Quando um agente é iniciado, um canal de comunicação entre ele e ODE é criado, permitindo que o agente acesse objetos de ODE por meio desse canal (um *socket*). Para agentes acessarem objetos, é usada a operação `executarMetodoObjeto` da classe *AgCliente*. O método do objeto a ser executado é passado como uma *string* e a resposta a esse método, caso não seja nula, é passada ao agente como uma *string* XML (*Extensible Markup Language*). Ou seja, os retornos dos métodos dos objetos de ODE são convertidos em uma *string* XML do lado de ODE e passados pelo canal de comunicação, de modo que o agente, ao receber essa *string* em XML, a converte em objetos do domínio do problema. Optou-se por usar XML, por ser uma linguagem padrão com marcações que permitem adicionar informação sobre o conteúdo de uma mensagem, o que facilita o tratamento das mensagens pelos agentes.

Deve-se realçar que ODE é totalmente independente de agentes. Se um usuário de ODE decidir que executará o ambiente sem o apoio de agentes, ODE terá apenas suas funcionalidades habituais e os agentes não serão iniciados. Caso ele decida usar agentes, ODE permitirá resolver seus problemas com a ajuda dos mesmos. Assim, somente os agentes vêem os objetos de ODE. O contrário não é verdadeiro. Agentes atuam dentro do ambiente sem que o mesmo tenha conhecimento disso. Os modos de execução do ambiente (com ou sem agentes) são escolhidos pelo usuário via configuração do ambiente.

4 Utilizando AgeODE

Algumas aplicações baseadas em agentes foram construídas para que a infra-estrutura desenvolvida pudesse ser avaliada, dentre elas a aplicação de controle de usuários de ODE, apresentada nesta seção para ilustrar o uso de AgeODE. Para a modelagem dessa aplicação, foi utilizada a metodologia OplA (*Objects plus Agents oriented Methodology*) [17], que oferece diretrizes para o desenvolvimento conjunto de agentes e objetos. A Figura 4 apresenta o diagrama de casos de uso dessa aplicação, onde o ator *Desenvolvedor* representa os diversos usuários de ODE (analistas, projetistas, programadores etc), enquanto o ator *Gerente de Projeto* é responsável por atividades referentes à gerência de projetos em ODE.

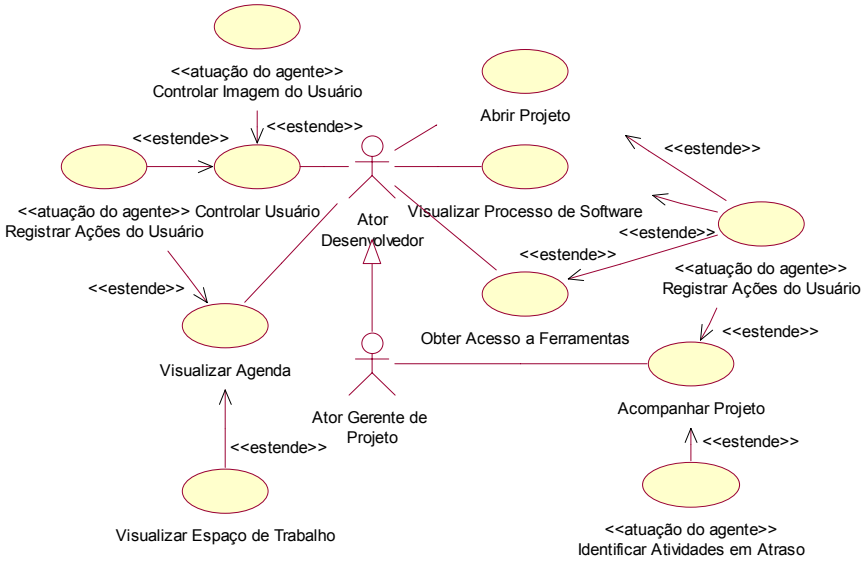

Fig. 4. Diagrama de Casos de Uso.

O caso de uso "Controlar Usuário" trata do acesso de usuários ao ambiente, por meio de *login* e senha. Ao acessar o ambiente, o desenvolvedor pode selecionar um projeto para trabalhar, realizando o caso de uso "Abrir Projeto". O desenvolvedor pode, então, visualizar sua agenda de trabalho (caso de uso "Visualizar Agenda"), a partir da qual lhe é permitido realizar atividades, podendo visualizar seu espaço de trabalho, isto é, seus artefatos, roteiros e normas de trabalho, apresentados de acordo com o projeto que está aberto e com a atividade selecionada (caso de uso "Visualizar Espaço de Trabalho"), além de registrar o esforço despendido na realização das mesmas. Finalmente, o desenvolvedor pode obter acesso a ferramentas de ODE que ele tem permissão de uso naquele momento e visualizar as atividades do processo de software do projeto, realizando, respectivamente, os casos de uso "Obter Acesso a Ferramentas" e "Visualizar Processo de Software".

O gerente de projeto, por sua vez, pode acompanhar os projetos de software sob sua responsabilidade, através do caso de uso "Acompanhar Projeto".

Os casos de uso com o estereótipo <<atuação do agente>> são aqueles tratados por agentes [17]. Os agentes de software identificados no contexto da aplicação têm como objetivo acompanhar o usuário que está utilizando o ambiente ODE, seja ele um desenvolvedor ou um gerente de projeto. Assim, dois agentes foram identificados: o Agente Assistente Pessoal (*AgAssistentePessoal*) e o Agente Gerente de Projeto (*AgGerenteProjeto*). Além desses dois agentes, há um agente do tipo roteador (*AgRoteadorODE*) para permitir a comunicação entre eles.

O agente assistente pessoal é um agente que agrega funcionalidades dos tipos agente de interface e agente de usuário, sendo o primeiro o tipo principal. Esse agente acompanha o usuário de ODE desde o momento que ele acessa o ambiente até o

momento em que ele sai, dando-lhe sugestões, avisos etc, de modo a facilitar seu trabalho, amenizar a sobrecarga de informações e auxiliá-lo em suas decisões.

Vale lembrar que, como os agentes são implementados em Java, uma linguagem que não suporta herança múltipla, os agentes só podem herdar de uma classe de agente, aquela que representa seu tipo principal, e devem implementar as interfaces de seus demais tipos. Assim, além das classes mostradas na figura 2, cada tipo de agente em AgeODE tem uma interface associada. No caso do *AgAssistentePessoal*, ele herda de *AgInterface* e implementa a interface *IAgUsuario*, como mostra a figura 5.

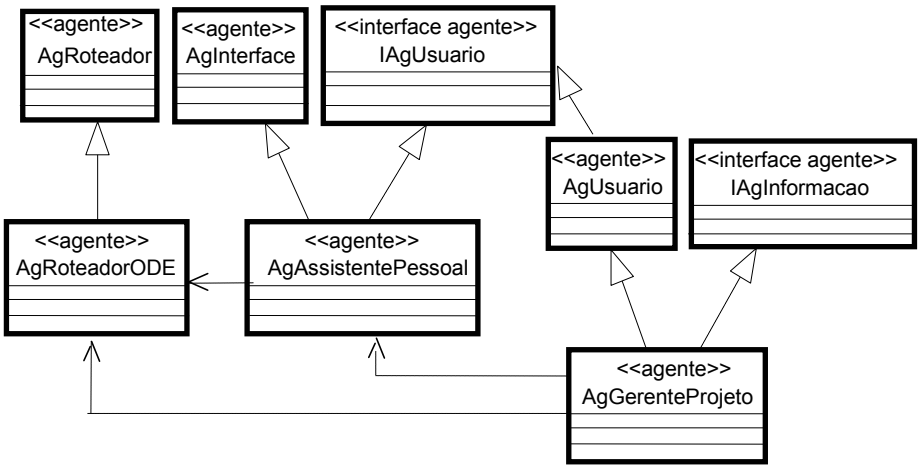

Fig. 5. Diagrama de Classes de Agentes.

Quando ODE é iniciado, o agente roteador é iniciado e, a seguir, o agente assistente pessoal é iniciado e se registra no roteador. Cada agente em ODE é identificado por um nome único e um endereço que são passados ao roteador no momento em que aquele se registra.

Quando um usuário se identifica em ODE, o "observador" do agente assistente pessoal percebe isso e o avisa para que ele possa capturar quem é o usuário que está entrando no ambiente. Isso ocorre também quando o usuário abre um projeto. Logo, o agente captura em qual projeto o usuário está trabalhando e se ele é o gerente desse projeto. Sendo ele o gerente de projeto, o agente assistente pessoal inicia o agente gerente de projeto, que, por sua vez, se registra junto ao roteador e envia uma mensagem ao agente assistente pessoal perguntando qual é o projeto e quem é o gerente.

As dependências entre os agentes mostradas na figura 5 indicam que tanto *AgAssistentePessoal* como *AgGerenteProjeto* dependem de *AgRoteador*, já que eles precisam se registrar nesse agente para poderem se comunicar. Além disso, *AgGerenteProjeto* depende de *AgAssistentePessoal*, uma vez que se comunica com esse agente para saber qual o projeto corrente e o usuário gerente de projeto.

O agente assistente pessoal é responsável, ainda, por salvar o *log* de ações do usuário, ou seja, ele guarda o histórico de utilização de ODE por parte daquele usuário (por exemplo, o usuário abriu o projeto X, abriu sua agenda, selecionou a atividade Y, realizou-a na ferramenta Z etc). Quando o usuário encerra uma sessão de

ODE deixando algum(ns) artefato(s) ou ferramenta(s) aberta(s), bem como, o projeto em que ele estava trabalhando, a imagem (configuração) do ambiente para esse usuário é guardada. Quando o mesmo usuário acessar novamente o ambiente, um resumo da imagem lhe é apresentado e lhe é perguntado se deseja que essa imagem seja restaurada, não necessitando esforço para se restabelecer o contexto no qual parou o seu trabalho da última vez, como mostra a figura 6. Constitui uma imagem do usuário no ambiente: a data do último acesso, o projeto em que ele trabalhou, bem como a(s) atividade(s) que ele estava desenvolvendo ao fechar o ambiente e o(s) artefato(s) e a(s) ferramenta(s) que estavam em uso.

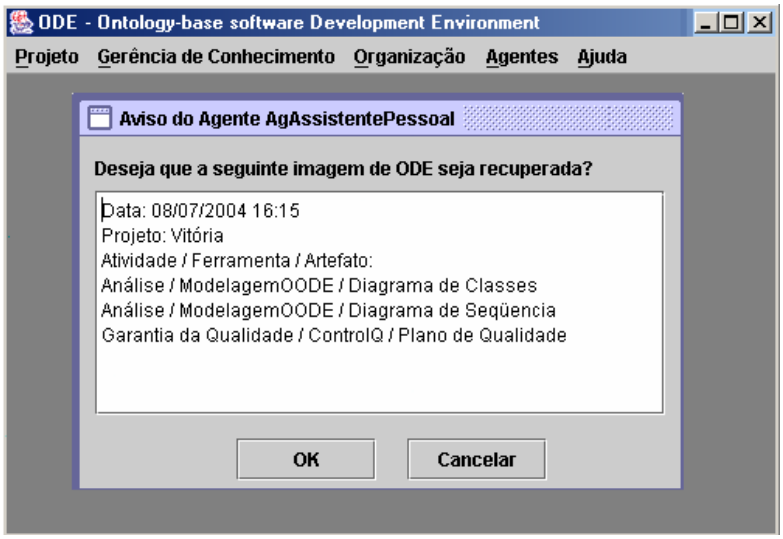

Fig. 6. Recuperando a imagem de ODE para o usuário.

O agente gerente de projeto, por sua vez, é um agente com funcionalidades dos tipos agente de informação e agente de usuário, sendo o último seu tipo principal. Esse agente auxilia o gerente de projeto no acompanhamento do projeto, auxiliando-o em algumas ações relativas à gerência de projetos, tal como a verificação de quais atividades estão em atraso no projeto.

Finalmente, vale ilustrar como se dá a comunicação entre agentes: ao ser iniciado, *AgGerenteProjeto* pergunta a *AgAssistentePessoal* qual o projeto aberto. Esta mensagem é escrita em KQML conforme a seguir:

```
ask-one
   :sender AgGerenteProjeto
   :receiver AgAssistentePessoal
   :language XML
   :ontology Processo
   :content <Ode.Controle.Cdp.Projeto/>
```

Na seqüência, *AgAssistentePessoal* responde, enviando a seguinte mensagem:

```
reply
   :sender AgAssistentePessoal
   :receiver AgGerenteProjeto
```

```
:language XML
:ontology Processo
:content <Ode.Controle.Cdp.Projeto id="2:101"/>
```

O conteúdo das mensagens (*:content*) é escrito em XML. No caso das mensagens requisitando objetos, como a mensagem do *AgGerenteProjeto*, deve-se descrever o objeto a ser passado com o caminho completo da classe a que pertence. Já no caso das mensagens informando objetos, como a resposta do *AgAssistentePessoal*, o conteúdo deve informar o identificador do objeto correspondente, de modo que o agente, ao receber e interpretar a mensagem, possa obter o objeto em questão por meio de seu identificador.

Após projetar os agentes, pode-se utilizar algumas facilidades oferecidas por AgeODE para implementá-los. É muito importante que todos os agentes de ODE sejam cadastrados no ambiente, pois alguns necessitam conhecer outros para o bom funcionamento do ambiente. Seja o exemplo do *AgAssistentePessoal*. Ele deve conhecer todos os agentes que atuam em ferramentas de ODE, para poder iniciá-los quando a respectiva ferramenta for iniciada. Caso um agente que atue em uma ferramenta não esteja cadastrado no cadastro de agentes de ODE, o *AgAssistentePessoal* não saberá de sua existência e, conseqüentemente, não poderá iniciá-lo. Assim, o cadastro de agentes de ODE serve, principalmente, de base de conhecimento sobre os agentes atuando em ODE, de modo que futuros desenvolvedores possam saber que agentes existem e para que os agentes de ODE se conheçam. Portanto, para um bom funcionamento do sistema multiagente de ODE, todos os agentes têm de ser cadastrados.

Uma vez que todos os agentes atuando em ODE têm de ser cadastrados usando a ferramenta de cadastro de agente, optou-se por estendê-la para apoiar, também, a construção de agentes em ODE, guiando o desenvolvedor na realização dos passos necessários para implementar um agente em AgeODE, gerando parcialmente o código dos agentes e registrando-os no Cadastro de Agentes de ODE. Desta forma, durante o cadastro de um agente, devem ser informados: nome e descrição do agente, qual(is) o(s) tipo(s) de agentes que o agente em questão especializa (indicando o principal) e em que ferramenta do ADS o agente atua. Além disso, deve-se indicar os objetos que compõem sua base de conhecimento (tratados como atributos), suas atividades (computações que podem ser executadas pelo agente sem interagir com outros agentes, que serão tratados como métodos da classe de agentes) e seus protocolos, que definem como um agente pode interagir com outros agentes, informando nome, descrição, o agente receptor da mensagem, a performativa e, opcionalmente, o conteúdo da mensagem. Após esse cadastro, o desenvolvedor pode pedir para que o código do agente seja gerado. Caso o agente seja do tipo interface, o código do objeto "observador" também é gerado.

5 Conclusões

A tecnologia de agentes é bastante aplicável na construção de Ambientes de Desenvolvimento de Software (ADSs) e, portanto, é importante ter uma infra-estrutura que apóie a construção de agentes para atuarem nesse contexto. Entretanto, as infra-estruturas existentes são de caráter geral, não levando em conta características

de agentes atuando em um ADS. Assim, foi desenvolvida AgeODE, uma infra-estrutura desse tipo, integrada ao ambiente ODE. A infra-estrutura foi aplicada para implementar algumas funcionalidades desse ambiente usando agentes, dentre elas as apresentadas neste artigo.

Com base no relato de desenvolvedores que utilizaram AgeODE, é possível levantar alguns pontos fortes e fracos da infra-estrutura. Dentre os pontos fortes, destacam-se:

- AgeODE favorece a padronização na construção de agentes e o reuso, uma vez que disponibiliza um *framework* de classes para a construção dos agentes;
- Os agentes clientes, além de poderem ser de um dos quatro tipos diferentes de agentes clientes oferecidos por AgeODE nesta versão do trabalho, podem implementar interfaces dos outros tipos, assumindo, assim, diversos perfis diferentes, uma vez que pode haver a combinação desses tipos;
- O acesso a objetos de ODE, por meio de *sockets*, é transparente para o desenvolvedor, bastando apenas executar um método passando o nome do método, seus parâmetros e o objeto sobre o qual será aplicado;
- Os agentes de interface construídos usando AgeODE são capazes de observar o ambiente, capturando as ações que os usuários realizam na interface com o usuário, mesmo estando os agentes em processos separados de ODE, de forma totalmente autônoma;
- AgeODE permite a automatização parcial da construção dos agentes, uma vez que é possível gerar parcialmente seu código, o que agiliza e facilita o trabalho do desenvolvedor e, conseqüentemente, padroniza o código dos agentes.

Entretanto, AgeODE também tem pontos fracos, a saber:

- A comunicação entre agentes é centralizada num agente roteador, sobrecarregando-o;
- O(s) objeto(s) que um agente acessa de ODE não são passados como objetos de fato ao agente, mas sim são passados seus identificadores e as classes a que pertencem, como uma *string* XML, via *socket*, sendo necessário que o agente acesse o banco de dados para recuperar o objeto;
- Cada agente que necessita acessar objetos de ODE, por rodar em um processo separado da aplicação principal do ambiente, tem que ter sua própria conexão com o banco de dados, o que sobrecarrega o banco de dados, porque em um dado momento pode haver vários agentes com uma conexão ativa;
- O desenvolvedor de um agente que necessite acessar o método de um objeto de ODE deve saber e codificar corretamente o caminho completo da classe do objeto, bem como o nome do método a ser acessado.

Como perspectiva futura, espera-se tratar os pontos fracos detectados, além de explorar novos tipos de agentes e ampliar a infra-estrutura proposta, permitindo tratar agentes inteligentes e agentes móveis. Para tal, facilidades de inferência têm de ser incorporadas a AgeODE.

Agradecimentos

Este trabalho foi realizado com o apoio do CNPq, uma entidade do Governo Brasileiro dedicada ao desenvolvimento científico e tecnológico.

Referências

1. Travassos, G. H.: O Modelo de Integração de Ferramentas da Estação TABA, Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 1994.
2. Falbo, R. A., Natali, A. C. C., Mian, P. G., Bertollo, G., Ruy, F. B.: ODE: Ontology-based software Development Environment. IX Congreso Argentino de Ciencias de la Computación, p. 1124-1135, La Plata, Argentina, Outubro 2003.
3. Wooldridge, M.: Intelligent Agents. In: Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence, London, The MIT Press, p. 27-77, April 1999.
4. Natali, A. C. C., Falbo, R. A.: Gerência de Conhecimento em ODE. Anais do XVII Simpósio Brasileiro de Engenharia de Software, p. 270-285, Manaus, Brasil, Outubro 2003.
5. Jeon, H., Petrie, C., Cutkosky, M. R.: JATLite: A Java Agent Infrastructure with Message Routing. IEEE Internet Computing, March /April 2000.
6. Bellifemmine, F., Poggi, A., Rimassa, G.: JADE - A FIPA2000 Compliant Agent Development Environment. 5th International Conference on Autonomous Agents, Montreal, Canadá, 2001.
7. Silva, F. A. D., Reis, R. Q., Reis, C. A. L., Nunes, D. J.: Um Modelo de Simulação de Processos de Software Baseado em Agentes Cooperativos. XIII Simpósio Brasileiro de Engenharia de Software, p. 163-178, Florianópolis, Santa Catarina, Brasil, 1999.
8. Murta, L. G. P. CHARON: Uma Máquina de Processos Extensível Baseada em Agentes Inteligentes. Dissertação de Mestrado, COPPE/UFRJ, Rio de Janeiro, Brasil, 2002.
9. Falbo, R. A., Ruy, F. B., Pezzin, J., Dal Moro, R.: Ontologias e Ambientes de Desenvolvimento de Software Semânticos. 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering, JIISIC'2004, Madrid, Spain, November 2004.
10. Falbo, R. A., Menezes, C. S., Rocha, A. R. C.: A Systematic Approach for Building Ontologies. Proceedings of the 6th Ibero-American Conference on Artificial Intelligence, Lisbon, Portugal, Lecture Notes in Computer Science, vol. 1484, 1998.
11. Ruy, F. B., Bertollo, G., Falbo, R. A.: Knowledge-based Support to Process Integration in ODE. Clei Electronic Journal, Volume 7, Number 1, June 2004.
12. Aridor, Y., Lange, D. B.: Agent Design Patterns: Elements of Agent Application Design. 2nd International Conference on Autonomous Agents, Minneapolis, May 1998.
13. Kendall, E. A., Krishna, P. V. M., Pathak, C. V., Surech, C. B.: Patterns of Intelligent and Mobile Agents. 2nd Int. Conference on Autonomous Agents, Minneapolis, May 1998.
14. Brugali, D., Sycara, K.: Towards Agent Oriented Application Frameworks. ACM Computing Surveys, Volume 32, Issue 1, March 2000.
15. Juchem, M., Bastos, R. M.: Projetando Sistemas Multiagentes em Organizações Empresariais. XVI Simpósio Brasileiro de Engenharia de Software, Gramado, Brasil, 2002.
16. Finin, T., Labrou, Y., Mayfield, J.: KQML as an agent communication language. September, 1995. Disponível em http://www.cs.umbc.edu/~finin/papers/
17. Schwambach, M. M., Pezzin, J., Falbo, R. A.: OplA: Uma Metodologia para o Desenvolvimento de Sistemas Baseados em Agentes e Objetos. 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering, JIISIC'2004, Madrid, Spain, November 2004.