UNIVERSIDADE DE CAXIAS DO SUL
CENTRO DE CIÊNCIA DA COMPUTAÇÃO E TECNOLOGIA DA INFORMAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

SIDINEI DANIEL LUBENOW

Gestão de perfis em Comunidades de Prática em Sistemas Multiagentes

Prof. Joao Luis Tavares da Silva
Orientador

Caxias do Sul, Dezembro de 2010

"A imaginação é mais importante
que o conhecimento.
O conhecimento é limitado.
A imaginação envolve o mundo."
— Albert Einstein

AGRADECIMENTOS

Aos meus pais, Mario e Marinês, que estiveram sempre do meu lado, com o seu apoio, conselhos, pela educação e forma que conduziram todo a aprendizagem e a noção do quanto importante é o ensino.

A minha namorada Mariane pelo apoio e compreensão nessa fase de desenvolvimento do tcc e por estar sempre ao meu lado.

Ao meu orientador João pelo apoio, paciência e auxílio nesse trabalho que proporcionou um grande desafio e oportunidade de desenvolver esse trabalho.

Aos amigos pela paciência com minha ausência e falta de tempo.

A todos os professores que contribuíram para que fosse possível subir mais um degrau e proporcionar a oportunidade para ser um profissional e uma pessoa melhor.

A todos da minha família que se importam e estiveram presentes nessa etapa.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

SMA	Sistema MultiAgente
JADE	Java Agent Development Framework
JAVA	Linguagem de programação orientada a objetos
OTICSSS	Observatório das Tecnologias de Informação e Comunicação em Sistemas e Serviços de Saúde
CVP	Comunidade Virtual de Prática
Middleware	Programa de computador que faz a mediação entre software e as demais aplicações.

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este trabalho descreve o estudo, análise, modelagem e implementação de agentes de informação que visam a gestão e partilha de conhecimento em um contexto de comunidades virtuais. O principal objetivo é definir uma plataforma descentralizada de gestão de perfis de usuários e comunidades usando sistemas multiagentes na definição de Comunidades de Prática. Foram modelados dois tipos de agentes, um que representa os comportamentos necessários a uma comunidade e outro que representa os usuários participantes nestas comunidades. Esses comportamentos armazenam o perfil com o qual é possível realizar a recomendação de agentes de acordo com a similaridade de interesse e conhecimento. Cada agente representa os interessese e conhecimento de um usuário e o perfil é privado somente sendo compartilhados informações específicas de uma área com os outros agentes no momento da análise à procura de intersecções de interesses para recomendações.

Palavras-chave: Agentes de Informação, Assistente Pessoal, Compartilhamento de conhecimento, Sistemas Multiagentes, JADE, Prometheus.

knowledge sharing in communities of practice using Multiagent Systems

ABSTRACT

This work describes the study, analysis, modeling and implementation of information agents aiming to manage and share knowledge in a context of virtual communities. The main goal is to define a management decentralized platform of user and communities profiles by using multi-agent systems in the definition of Communities of Practice. We have modeled two types of agents, one representing the behaviors needed to the community and other representing the users participating in the community. These behaviors store the profile wich make possible to agents take some recommendation according to the similarity of their interest and knowledge. Each agent represents the knowledge of the user and this profile is private, thus only a specific part of the knowledge with the other agents is shared, when occur the analysis looking for intersections of interest for recommendations.

Keywords: Information Agents, Personal Assistant, Sharing knowledge, Multiagent Systems, JADE, Prometheus.

1 INTRODUÇÃO

Atualmente a noção de comunidade constitui-se em um dos focos principais de atenção em Ciências Sociais, Administração, Economia, Ciências Políticas e, relacionado às comunidades virtuais, também no desenvolvimento de Software Livre. A partir do crescente desenvolvimento da tecnologia de portais corporativos e ferramentas eletrônicas de comunicação, as organizações tendem a explorar o uso das tecnologias colaborativas para desenvolver novas capacidades organizacionais (DUBé; BOURHIS; JACOB, 2005). Segundo Wenger e Snyder (WENGER; MCDERMOTT; SNYDER, 2002), uma das maneiras mais eficientes de desenvolver estas capacidades organizacionais é através da Gestão do Conhecimento baseado no trabalho colaborativo. Neste contexto, grupos de pessoas trabalham juntos e aprendem com as outras e com o conhecimento compartilhado, propiciando um modo inovador de combinar trabalho, aprendizagem e inovação.

Nesta nova perspectiva, o uso das comunidades assume um papel relevante em termos de aprendizado e manutenção do conhecimento comum a um grupo de pessoas e mesmo a uma organização. Um dos grandes interesses gerais na aprendizagem organizacional hoje em dia, encontra-se em conjuntos de pessoas que se reúnem por um interesse comum em aprender e aplicar práticas comuns de forma espontânea, compartilhando e absorvendo conhecimento e experiência com outras pessoas. Esta ideia define o que Etienne Wenger (WENGER, 2008) chamou de *Comunidades de prática (CdP)*, que também podem ser informalmente caracterizadas como grupos de pessoas que compartilham um conceito ou uma paixão por algo que elas fazem e que interagem regularmente para aprender a fazê-lo melhor.

O que precisa ser identificado nas CdPs é o perfil dos participantes e mapear o conhecimento intra-comunidade e inter-comunidade para diagnosticar interseções entre os usuários. Com essa identificação é possível sugerir e informar outros que tem conhecimentos em comum. Com a possibilidade de solução de mapeamento destes perfis usando metodologias descentralizadas e distribuídas podemos utilizar um sistema multiagentes, onde cada usuário da comunidade tem seu próprio agente.

Nesta abordagem, cada usuário é representado por um agente de software que

gerencia o perfil do usuário em termos de crenças e conhecimentos que o mesmo possui, seja ele individual ou de acordo com as comunidades que o mesmo está inserido. O agente poderá se comunicar e negociar com outros agentes com base nas crenças que possui e sugerir ao usuário outros relacionamentos que possuem um nível de ontologia comum. A ontologia é uma ontologia de domínio, definida pela comunidade e especificada no projeto Perf-CdP (institucional UCS), não cabendo ao agente a responsabilidade de adicionar ou modificar a mesma. O principal objetivo do Projeto Perf-CdP é o desenvolvimento de um framework computacional para gestão de Comunidades de Prática (CdP), onde os perfis de interesse dos usuários e das próprias comunidades sejam gerenciados por agentes virtuais, sendo que cada agente virtual representa os interesses de um usuário.

A colaboração entre vários agentes representando membros da comunidade, juntamente com um mecanismo de gestão da própria comunidade, necessitam de um mapeamento deste conhecimento. O levantamento do perfil de interesse irá proporcionar o mapeamento das especializações do usuário e possibilitar a seleção daquelas que preenchem os requisitos definidos. Por fim, estas interações multiagentes definem a adoção de critérios que fazem convergir as referências aos requisitos dos usuários a partir da utilização de uma ontologia de domínio que contemple os conhecimentos veiculados na CdP. Esta funcionalidade define potenciais membros para outras comunidades, a relação de comunidades que interessam a determinados usuários e também a recomendação de expertises de usuários no contexto de um problema específico. Estes mapas de especialização visam o compartilhamento de conhecimento tácito e o desenvolvimento de comunidades, ao permitir que as pessoas encontrem e estabeleçam contato pessoal mais rapidamente umas com as outras. Perfis de usuários detalhados, precisos, atualizados e significativos ajudam a fomentar conexões e elevar o nível necessário de confiança entre os participantes.

1.1 Objetivos e Motivações

Neste trabalho de conclusão pretende-se fazer a especificação, modelagem e desenvolvimento de um agente de informação para compartilhamento de conhecimento em Comunidades de Prática. Neste contexto, os perfis de interesse dos usuários e das próprias comunidades serão gerenciados por agentes virtuais, sendo que cada agente virtual representa os interesses de um usuário. A implementação de um protótipo baseado em comunidades de agentes objetiva a gestão da partilha de conhecimento entre os membros de Comunidades de Prática que mais tarde poderá ser agregado ao portal do Observatório OTICS.

Os objetivos específicos desse trabalho são:

1. Estudo do estado-da-arte sobre: sistemas multiagentes, agentes de informação,

ontologias como suporte à informação;

2. Especificação de requisitos para o agente a partir de uma metodologia AOSE;
3. Modelagem e implementação do comportamento do agente; especificação de um ambiente multiagente para simulação, utilizando uma ontologia de domínio;
4. Implementação do sistema multiagentes responsável pela gestão do perfil de conhecimento da CdP;
5. Estudo de caso: modelagem da gestão do Perfil dos participantes e o Perfil da própria CdP através de uma simulação multiagentes;
6. Experimentos.

1.2 Organização do Texto

No próximo Capítulo será apresentada a fundamentação teórica detalhando os conceitos fundamentais de Comunidades de Prática, Sistemas Multiagentes e as metodologias e plataformas para sua especificação e implementação. No Capítulo 3, será apresentada a modelagem e especificação da solução proposta, utilizando a metodologia Prometheus. A implementação da solução em JAVA e utilizando o Framework JADE será ilustrada no Capítulo 4 e a validação da solução com utilização dos cenários ilustrados é apresentando no Capítulo 5. Finalmente, as conclusões e perspectivas futuras são apresentadas no Capítulo 6.

2 FUNDAMENTAÇÃO TEÓRICA

Para a consecução deste trabalho, nos baseamos em definições da área de Gestão do Conhecimento quanto aos aspectos conceituais e análise do problema. Também utilisaremos metodologias da área de Sistemas Multiagentes para a implementação do protótipo de solução, onde os perfis dos usuários e das comunidades são representados por agentes virtuais. Neste Capítulo, apresentamos a fundamentação básica que orienta nossa proposta em termos de comunidades de prática e agentes de informação, bem como das metodologias e ferramentas necessárias para a implementação da solução computacional.

2.1 Comunidades Virtuais e Comunidades de Prática

Na "espiral do conhecimento" (NONAKA, 2001) define conhecimento em dois tipos distintos: explícito e tácito, sendo explícito todo aquele conhecimento que pode ser facilmente formalizado ou sintetizado, enquanto o tácito está presente no indivíduo, e confunde-se ao seu modo de agir e ser, de tão enraizado em suas experiências. Neste sentido, Nonaka define a necessidade em transformar conhecimento tácito em explícito através de quatro formas de conversão: socialização (tácito em tácito); externalização (tácito em explícito); combinação (explícito em explícito); e, internalização (explícito em tácito).

A socialização do conhecimento pode ocorrer de diversas maneiras e meios, entre estes estão as Comunidades de Prática (CdP). O uso das comunidades assume um papel relevante em termos de aprendizado e manutenção do conhecimento comum a um grupo de pessoas e mesmo a uma organização. Um dos grandes interesses gerais na aprendizagem organizacional, hoje em dia, encontra-se pessoas que reúnem-se para aprender e aplicar os interesses que possuem em comum, compartilhando e absorvendo conhecimento e experiência com outras pessoas. Etienne Wenger chamou de Comunidade de Prática grupos de pessoas que compartilham um conceito ou uma paixão por algo que fazem e que interagem regularmente para aprender a fazê-lo melhor com interações regulares(WENGER, 2008). Essa interação entre os membros

pode ser de forma virtual ou presencial desde que possibilite a troca de conhecimento auxiliando na busca de soluções e melhores práticas para promover o aprendizado em grupo.

As comunidades apresentam características fundamentais que segundo (WENGER, 2008) são resumidos em três grupos: domínio, comunidade e prática. O domínio é o tema ou assunto sobre o qual os membros da comunidade compartilham informações e experiências. A comunidade é a participação dos membros se ajudando mutuamente através de discussões e atividades promovendo o compartilhamento de informações e experiências. E a prática é a participação dos membros com a experiência de vida e resolução de problemas do grupo atráves do conhecimento adquirido.

Já para (KOCH; LACHER, 2005), Comunidade é definido como um grupo de pessoas que compartilham o mesmo interesse ou estão inseridas em um mesmo contexto.

De forma geral, pode-se definir comunidade como sendo um grupo de pessoas que compartilham os mesmos propósitos de forma a permitir e/ou auxiliar a resolução de problemas, ou seja, grupos de pessoas e/ou profissionais com interesses e/ou trabalhos similares. Logo, percebe-se que os elementos básicos que formam qualquer comunidade são os indivíduos, as maneiras como eles se relacionam e o contexto ou domínio que estes estão inseridos.

O termo Comunidade Virtual pode ser visto como um ambiente onde pessoas com interesses comuns trocam informações umas com as outras através de uma rede on-line, como a Internet. Através dessas trocas de informações, os participantes desenvolvem ligações com outros membros da comunidade e com a comunidade como um todo. Um aspecto importante sobre as Comunidades Virtuais é o aspecto da escalabilidade, pois a quantidade de membros de uma comunidade é dinâmico e variável, podendo os membros se filiarem e desfiliarem quando sentirem necessidade.

O que precisa ser identificado nas CdPs é o perfil dos participantes e o mapeamento do conhecimento intra-comunidade e inter-comunidade para diagnosticar interseções entre os usuários. Com essa identificação é possível recomendar e informar conhecimentos em comum entre eles. Com a possibilidade de solução de mapeamento destes perfis usando metodologias descentralizadas e distribuídas podemos utilizar um sistema multiagentes, onde cada usuário da comunidade pode ser representado por seu próprio agente. Alguns trabalhos refletem esta necessidade e possibilidade de aliar a tecnologias de agentes ao uso de conhecimento semântico na recuperação de informações.

O trabalho de Lugo (LUGO, 2004), por exemplo, propõe que o compartilhamento de informações em comunidades virtuais pode ser feito sobre demanda de objetivos, com cooperação entre agentes. As informações são ranqueadas utilizando

o conteúdo de documentos e e-mails. Esse conteúdo é utilizado como base para o estabelecimento de relações entre os agentes e o conhecimento do usuário é influenciado pela quantidade de vezes que cada palavra chave aparece nesses documentos sendo considerado o grau ou relevância que o usuário tem na área.

A estruturação desses dados é feita com base em ontologia e o conhecimento está relacionado e dependente da experiência que cada agente acumula. Os contatos com outros usuários serão organizados em grupo ou papel pelo usuário. Os conceitos da ontologia possuem termos associados que são considerados atributos. Os atributos são utilizados para indexar documentos e e-mails. Os e-mails são utilizados para determinar os sub-grupos contendo relacionamentos entre si que já possuem conhecimentos combinados. Quanto mais contatos um grupo de usuários possuir em suas relações, mais próximos eles estarão.

Outro trabalho nesta linha apresenta um suporte às comunidades para torná-las mais efetivas através de um ambiente de gestão do conhecimento para Comunidades Virtuais, o OntoShare(DAVIES; DUKE; SURE, 2004). Esta abordagem utiliza recursos semânticos em um ambiente proposto para suportar comunidades virtuais e faz uso das tecnologias da Web Semântica nessa construção. Modela os interesses de cada membro em um perfil de usuário através de um conjunto de termos ou conceitos ontológicos que expressam os seus interesses. O OntoShare tem a capacidade de sintetizar e extrair palavras-chaves de páginas Web e de outras fontes de informação compartilhada por um usuário e, então, compartilhar essas informações com outros usuários da comunidade cujo perfil de usuário os aponte como possíveis interessados na informação.

Esse ambiente é usado para armazenar, recuperar, resumir e disponibilizar aos outros membros informações consideradas importantes para certos membros. Ele modifica o perfil do usuário baseado no uso do sistema pelo mesmo, na tentativa de refinar o seu perfil para que corresponda realmente ao seu interesse. Quando um usuário encontra informações de seu interesse para ser compartilhada em sua comunidade, uma requisição de compartilhamento é executada. Então, convida o usuário a criar uma anotação para ser armazenada junto com sua informação. Neste ponto, o sistema usa a ontologia da comunidade para sugerir os conceitos que o usuário pode utilizar para a anotação sobre a informação. Neste momento, o OntoShare tenta evoluir a ontologia da comunidade, baseada nas anotações feitas pelo usuário. Uma vez que o ambiente permite o armazenamento de informação, ele também permite o compartilhamento dessa informação com outros membros da comunidade.

A ideia apresentada neste trabalho propõe questões similares a este trabalhos. Não estamos trabalhando com uma informação previamente classificada na ontologia, como a proposta em (LUGO, 2004). Na solução proposta a repetição de termos

(ou qualificação enviada pelo portal) irá ajudar a determinar o grau de importância combinado com o valor(fator) da ontologia que o termo se encontra. E também a solicitação de recomendação pode ser solicitada pelo usuário, pela comunidade, ou semi-atomatizada segundo algum critério de atualização que possa ser definido segundo a comunidade. A recomendação será realizada através do conhecimento e similaridade de interesses, sem estabelecer uma relação de proximidade pela interação prévia entre os agentes. Para que qualquer Comunidade Virtual tenha sucesso é interessante que ela possa aumentar o número de indivíduos participantes sem perder o "senso de comunidade", ou seja, todos da comunidade devem ter os mesmos objetivos.

2.2 Sistemas Multiagentes

Sistemas Multiagentes (SMA) podem ser compreendidos como uma abstração da Inteligência Artificial Distribuída em que entidades computacionais são capazes de interagir de forma autônoma e inteligente em ambientes compartilhados. Segundo Wooldridge (WOOLDRIDGE, 2009), agentes são entidades reais ou artificiais capazes de interagir em um ambiente de forma organizada para resolverem problemas em coletivos ou individuais.

Devido ao conhecimento do ambiente e dos agentes envolvidos em determinado contexto ou tarefa de resolução de problemas, os agentes possuem capacidades de comunicação que habilitam a interação multiagentes. Os tipos de comunicação envolvidos podem ocorrer através de estruturas centralizadas, troca direta de mensagens ou percepção do ambiente. No caso de agentes cognitivos, inteligentes e de informação, ocorrem trocas de mensagens explícitas que habilitam o comportamento social entre os agentes. Segundo a FIPA (*Foundation for Intelligent Phisical Agents*), uma plataforma de agentes deve fornecer suporte para comunicação entre agentes através de uma linguagem de comunicação, especificação de protocolos e serviços de transporte e gestão de conteúdo. A FIPA é uma organização de normas da IEEE Computer Society que promove a tecnologia baseada em agentes e a interoperabilidade de seus padrões com outras tecnologias.

Uma sociedade de agentes, constitui-se em um grupo formado estática ou dinamicamente para resolver um problema comum, onde um único agente não é capaz de solucionar. Existem várias definições e modelos de organização de agentes em sistemas multiagentes que definem metodologias específicas de resolução de problemas. Grande parte dos modelos organizacionais de agentes derivam de metáforas sociais do comportamento humano. Modelos e conceitos emprestados da Administração e Ciências Sociais, além da Psicologia, definem comportamentos individuais e coletivos dos agentes em nível social. No contexto de comunidades, a organização multiagente

é um componente fundamental para modelagem dos comportamentos artificiais, servindo como suporte analítico para verificação comportamental do agente em relação aos outros componentes (humanos ou virtuais) de uma comunidade.

Uma tipologia de agentes proposta por Nwana (1996), apresenta agentes do tipo colaborativos, competitivos, de informação, de interface, móveis, estacionários, híbridos, reativos, autônomos e inteligentes. Os tipos de agente que melhor concernem ao projeto atual são os agentes de informação, que procuram localizar, manipular e ordenar as informações provenientes de várias fontes distribuídas de forma autônoma, tornando-as acessíveis para outros sistemas ou usuários. Agentes de informação são softwares que acessam dados em diversas bases e podem estar distribuídas fisicamente. Também podemos chamar de agentes cooperativos e autônomos que solucionam objetivo em comum. Esses agentes consultam, analisam e integram dados de diferentes fontes de informação. Segundo Russel e Norvig (RUSSELL S. J.; IORVIG, 1995) os Agentes são divididos em grupos de acordo com a inteligência programada.

Agentes de informação são capazes de acessar múltiplas fontes de dados que são distribuídos e heterogêneos. Também são capazes de mediar e manter informações relevantes sobre representação de usuários ou de outros agentes de uma forma próativa. Podendo ser cooperativos ou não, estes devem ser racionais a fim de decidir como e quando executar tarefas e como medir o benefício destas ações em relação ao problema a ser solucionado.

Na concepção de (WOOLDRIDGE, 2009) um agente é uma entidade com capacidade de resolução de problemas encapsulados e possui as propriedades de:

- reatividade, capacidade de manter a interação com o ambiente e reagir a mudanças, como, por exemplo, observar e realizar ações no mundo;
- pró-atividade, capacidade de tomar iniciativa, caracterizando um comportamento orientado a objetivos tomando a iniciativa para atingí-lo;
- habilidade social, ser capaz de realizar ações sociais como comunicação e cooperação, para completarem a resolução de seus problemas, seja ele o auxílio de outros agentes (humanos ou computacionais) ou a decisão de interações apropriadas. Comunica-se com outros agentes e inclusive com seres humanos através de uma linguagem comum;
- autonomia, possibilidade de operar sem a intervenção direta de outros agentes (possivelmente humanos), e controlar totalmente suas ações e estado interno.

Na próxima seção trataremos de uma metodologia para modelagem de sistemas multiagentes, utizada na presente proposta e apresentaremos a plataforma JADE de desenvolvimento de sistemas multiagentes.

2.3 Metodologia Prometheus

A metodologia Prometheus (PADGHAM; WINIKOFF, 2004) apresenta um processo detalhado de especificação, design, implementação e testes de software orientado a agentes. Prometheus define uma linguagem de modelagem que é genérica a qualquer arquitetura e ambiente de implementação de SMA. A metodologia é composta de três fases, das quais são realizados artefatos gráficos e textuais estruturados. Estas três fases compreendem componentes produzidos que são utilizados desde a geração de esqueleto do código até a realização de testes. A Figura 2.1 mostra a organização da metodologia.

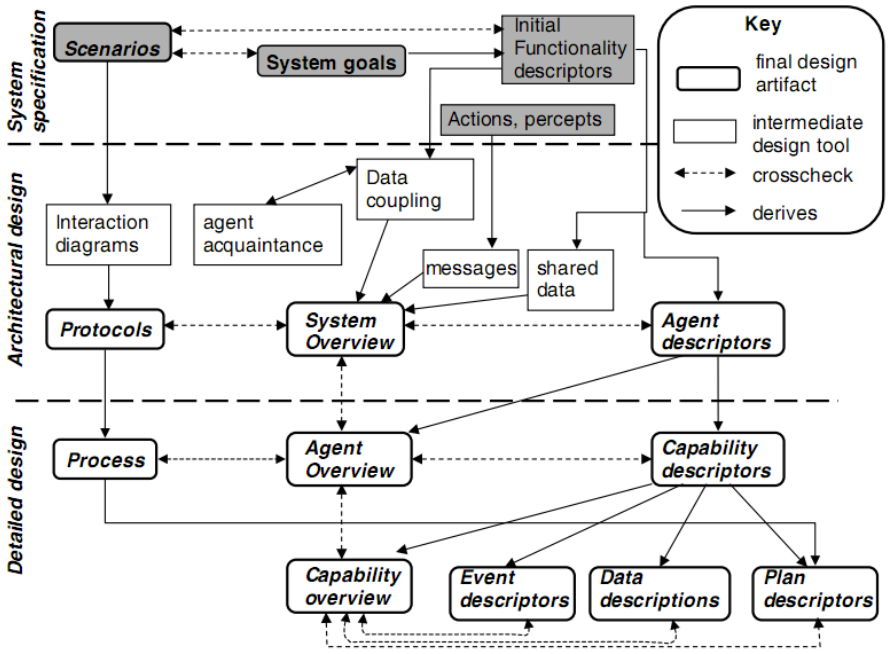

Figura 2.1: Fases da metodologia Prometheus(PADGHAM; WINIKOFF, 2004)

2.3.1 Especificação do Sistema

A Especificação do Sistema tem como objetivo identificar as funcionalidades e objetivos do sistema. Os cenários de caso de uso, as entradas de informação, as metas para a resolução do problema, os papéis dos agentes e o estabelecimento da interface entre os Sistema e o Ambiente de acordo com a relação das ações e percepções são itens definidos nessa fase.

Os cenários contêm a sequência de passos que são executados em um caso de uso, as ações são os resultados obtidos como resultado de uma funcionalidade a partir da entrada (percepção). Entre a entrada e saída as operações que são executadas são metas e sub-metas de acordo com um plano de execução.

2.3.2 Desenvolvimento da Arquitetura

A fase de desenvolvimento arquitetural utiliza o resultado da fase de especificação do sistema para determinar a interação entre os agentes. As atividades, definição dos tipos de agentes, definição da estrutura do sistema e definição das interações entre os agentes serão desenvolvidas nesta fase.

O diagrama de dados tem em seu modelo as informações necessárias para que cada agente execute os seus papéis, e para cada um que o agente participa é agrupado no diagrama de grupo de *Agent-Role*. O diagrama de visão geral do sistema contém o conteúdo da análise com a modelagem das bases de dados que cada agente acessa e obtém informações e a comunicação que é estabelecida entre os agentes atráves de protocolos.

2.3.3 Detalhamento do Projeto

A fase de detalhamento do projeto foca no desenvolvimento da estrutura interna de cada agente, como ele realizará sua tarefa dentro do sistema e os planos necessários para capacidade e o refinamento em definir as capacidades, planos e detalhes da estrutura de dados. Nesta fase é possível para cada agente envolvido no sistema definir suas capacidades, os eventos internos, os planos e a estrutura de dados detalhada de cada tipo de agente. A capacidade é detalhada em outro diagrama com a definição do plano de execução.

2.3.4 Diagramas Prometheus

Os componentes utilizados nos diagramas do Prometheus estão ilustrados na Figura 2.2. O componente *Agent* é utilizado para ilustrar os agentes envolvidos no sistema. O resultado produzido pelo agente e enviado para o ambiente é modelado pelo componente *Action*. Para conseguir produzir esse resultado, é necessário atingir metas *Goal* de acordo com o papel *Role* que ele tem no Sistema.

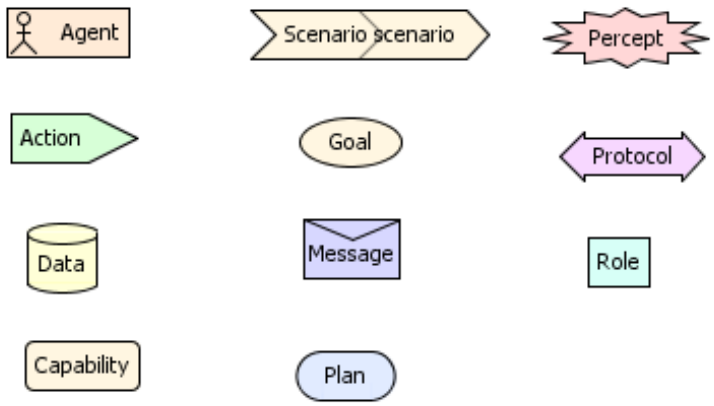

Figura 2.2: Componentes Utilizados Nos Diagramas

Os dados que o ambiente fornece ao agente provém das percepções (*Percept*). Em um sistema podemos ter um ou mais cenários(*Scenario*) e estes contém uma sequência abstrata de passos para a execução de um objetivo. Para conseguir atender a um objetivo em alguns casos é necessário a interação com outros agentes e essa é especificada por *Protocol*(protocolo) com a definição da sequência válida para a troca de mensagens, estas mensagens são modeladas com o componente (*Message*) e contém as informações de entrada e saída que são transmitidas.

Os dados acessados e manipulados pelo Agente estão representados pelo componente *Data*. Cada Agente tem uma capacidade específica no sistema e está representado pelo componente *Capability*. A sequência de ações e regras necessárias para que o agente atinja seu objetivo é especificada no plano modelado pelo componente *Plan*.

O diagrama da Figura 2.3 ilustra a especificação inicial para um sistema cuja funcionalidade é consultar músicas. Nesse caso o *AgenteConsulta* recebe do ambiente os dados de filtro e de login atráves da percepção (*Consulta*) e fornece ao ambiente a ação *Dados Musicas* com o resultado obtido. O diagrama da Figura 2.4 apresenta a base de dados ou crenças (*Musicas BD*) que o agente interage e o protocolo (*Comunicação*) com a definição de interação com outros agentes.

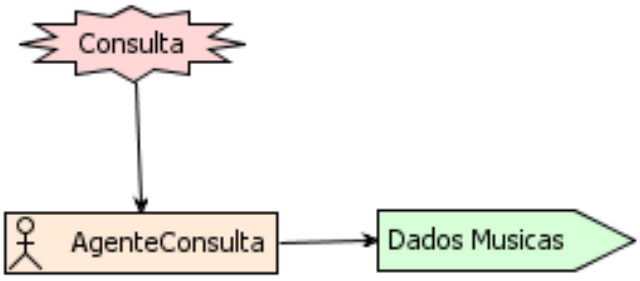

Figura 2.3: Exemplo Diagrama de Análise do Sistema

O papel que ele tem é *Consultar Musicas* e esse papel, além dos dados de entrada e saída, tem associado as metas *Valida Usuario*, *Valida Informação Filtro* e *ConsultaMusica*, conforme o modelo ilustrado na Figura 2.5.

O detalhamento das funcionalidades do agente está no diagrama da Figura 2.6, neste diagrama é definido a capacidade (*Capacidade Consultar Musicas*) com as relações de entrada, saída e dados que são necessários para esta capacidade.

Esta metodologia facilita o aprendizado em sistemas baseados em agentes, graças à flexibilidade no uso de vários tipos de agentes e possibilita também a validação e geração de um esqueleto de código.

Diferente de outras metodologias que apresentam muitas especificidades quanto

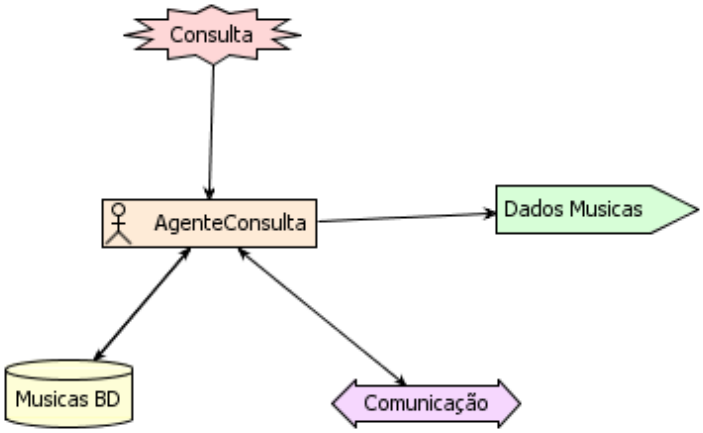

Figura 2.4: Exemplo Diagrama Sistema

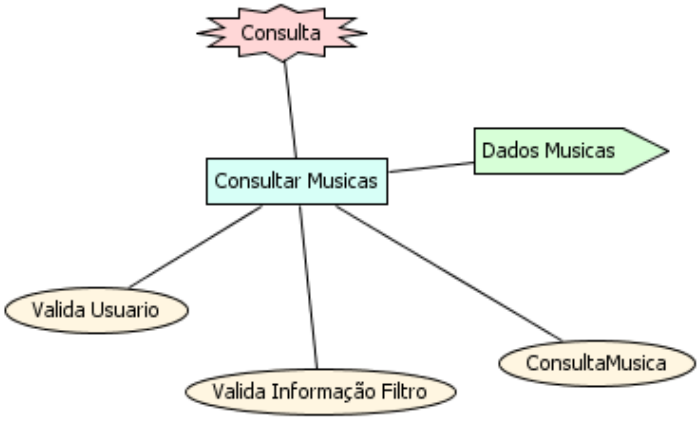

Figura 2.5: Exemplo de Papéis do Sistema

ao tipo de SMA ou aplicação a ser modelado. Por exemplo, a metodologia GAIA (ZAMBONELLI; JENNINGS; WOOLDRIDGE, 2003) também apresenta uma linguagem própria para a modelagem do SMA, definindo um processo de desenvolvimento em duas fases: análise e projeto. Na fase de análise, a metodologia GAIA coleta e organiza a especificação que constituirá a base para a fase de projeto. Entretanto, esta metodologia requer que o SMA seja estático quanto a quantidade de agentes, seus comportamentos, habilidades e as interações (BERNY; ADAMATTI; COSTA, 2009).

Já a metodologia Tropos (BRESCIANI et al., 2004) fornece suporte também às atividades de análise e projeto no desenvolvimento do SMA. A análise prevê a definição de ”stakeholders do domínio”, modelados como atores sociais. Para todos os atores definidos é necessário modelar um novo ator que representa o sistema e as dependências com os outros atores do ambiente (GIUNCHIGLIA; MYLOPOULOS; PERINI, 2002). Nesta metodologia, entretanto, conforme (SILVA; MARIA;

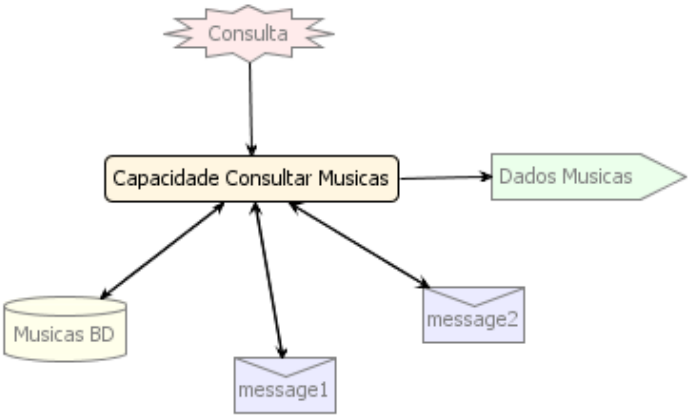

Figura 2.6: Exemplo Diagrama do Agente

LUCENA, 2006), a modelagem se mostra confusa e rebuscada, dificultando a fase do processo de desenvolvimento. E, também, a fase de projeto detalhado é orientada especificamente a plataforma JACK[1] (*Jack Intelligent Agents*), mas dificultando a abstração de pedaços genéricos de código, como é o caso do Prometheus, que permite uma separação melhor entre as partes de código próprias ao JACK.

2.4 Plataforma de Desenvolvimento JADE

O middleware (JADE, 2010) (BELLIFEMINE; POGGI; RIMASSA, 2001b; BELLIFEMINE et al., 2008), é baseado nas especificações da FIPA(*Foundation for Intelligent Physical Agents*), este auxilia no desenvolvimento de sistemas multiagentes desenvolvido em Java. A comunicação do agente é feita por troca de mensagens. Suporta a programação de comportamentos cooperativos, cíclicos e disparados por eventos sobre dados, resposta a mensagens e disparadores necessários para responder a uma solicitação. Tem suporte para linguagem de conteúdo definido pelo usuário e ontologias que podem ser implementadas e utilizadas pelo agente e automaticamente utilizadas pela estrutura.

A Figura 2.7 é a interface gráfica da plataforma JADE que permite gerenciar remotamente e acompanhar a execução, monitorar e controlar a qualidade dos agentes, criar e destruir os agentes.

A plataforma distribuída de agentes JADE pode ser dividida em vários ”hosts” ou máquinas (desde que eles possam ser conectados via RMI). Apenas uma aplicação Java e uma *Java Virtual Machine* é executada em cada host. Os agentes são implementados como *threads* Java e inseridos dentro de um container (*Agent Containers*) que provêm todo o suporte para a execução do agente.

[1]http://www.agent-software.com/products/jack/; acessado em out/2010

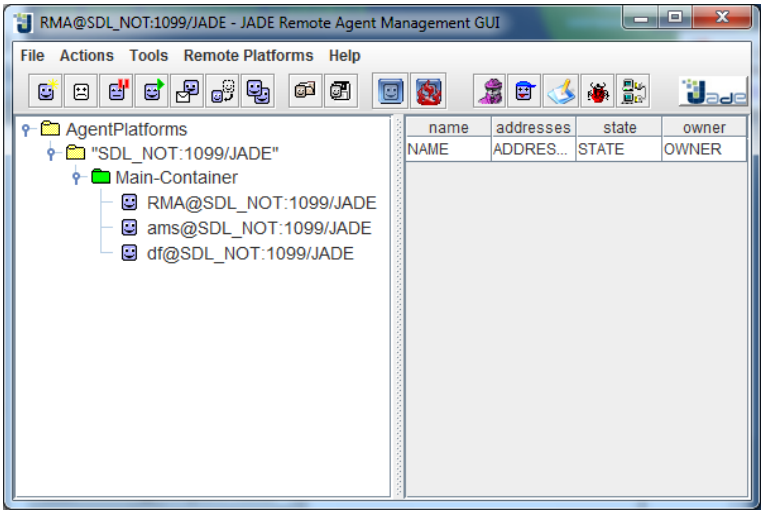

Figura 2.7: Interface para Gerenciamento de Agentes

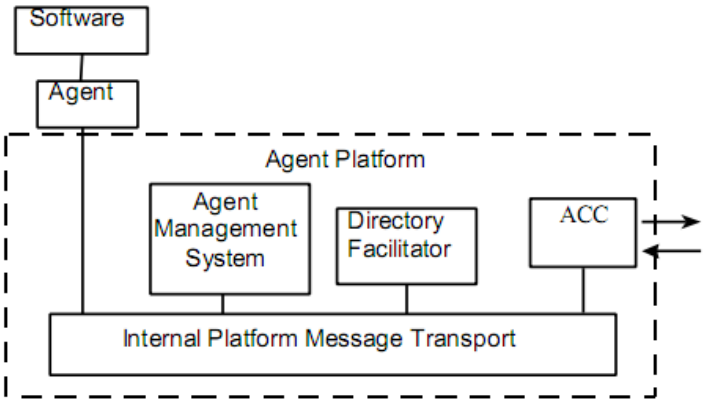

Figura 2.8: Plataforma Jade(BELLIFEMINE; POGGI; RIMASSA, 2001a)

O Ambiente de agentes inclui o sistema gerenciador de agentes AMS(*Agent Management System*), o diretório facilitador DF(*Directory Facilitator*) e o canal de comunicação de agentes ACC(*Agent Communication Channel*).

A organização da plataforma JADE está demonstrada na Figura 2.8. O AMS tem como finalidade manter os registros dos Agentes, os seus identificadores e estados. Todo agente precisa se registrar e na criação de um novo agente ele realizará o registro na AMS. O DF fornece o serviço de páginas amarelas para a plataforma, com serviços de registro, modificação e busca de agentes e serviços. Também é possível criar grupos de busca com propagação através de domínios e subdomínios.

2.5 Considerações Finais

Neste Capítulo relacionamos as principais referências teóricas dos quais nos utilizamos neste trabalho. Com o objetivo de definir um modelo de SMA que possa

definir o comportamento de troca de conhecimento entre comunidades de prática, foi definido agentes de informação como modelo para o comportamento dos agentes desta proposta, uma metodologia multiagentes que garantisse a especificação e documentação da solução e uma plataforma de desenvolvimento de sistemas multiagentes. O principal objetivo deste trabalho é fornecer a modelagem do SMA que represente agentes de usuários para manter seus perfis individuais e garantir a privacidade de suas informações. Deste modo, podemos relacionar o perfil de interesses de cada um dos participantes de uma comunidade de prática e definir uma camada de gestão de recomendação de comunidades e indivíduos através das interações entre os agentes dos usuários. No próximo capítulo usaremos a metodologia Prometheus para a especificação completa da solução proposta com a possibilidade de geração parcial de código utilizável na plataforma JADE, para a implementação de um protótipo para validação da solução.

3 ESPECIFICAÇÃO DO SISTEMA

No capítulo anterior foi ilustrado as técnicas e conceitos envolvidos na solução para o problema. Neste capítulo será descrita a especificação do agente de informação que irá compartilhar conhecimento em uma Comunidade de Prática. Para a especificação da solução será utilizado a metodologia Prometheus apresentada anteriormente. Será apresentado uma visão macro da solução e após o detalhamento das funcionalidades de cada agente assim como o plano de execução para cada capacidade que o mesmo possui.

3.1 Visão Geral

O Sistema Multiagente para atender aos requisitos e solucionar os objetivos propostos no trabalho foi analisado e as suas principais funcionalidades são:

- compartilhamento de conhecimento entre os agentes de informação de uma Comunidades de Prática;
- recomendação de agentes/usuários de acordo com a similaridade de interesses e conhecimento;
- mapeamento do conhecimento do usuário obtido pelas práticas e uso nas comunidades que participa em uma estrutura de dados que permita atráves do uso de um algoritmo de recomendação que encontre relações entre os agentes.

Na Figura 3.1 é demonstrado o modelo e etapas envolvidas na execução dos principais requisitos em que o Agente Usuário está envolvido. Neste modelo é representado a sequência de execução, validações, regras e planos para que a partir de uma solicitação seja possível retornar uma saída válida para o objetivo/funcionalidade.

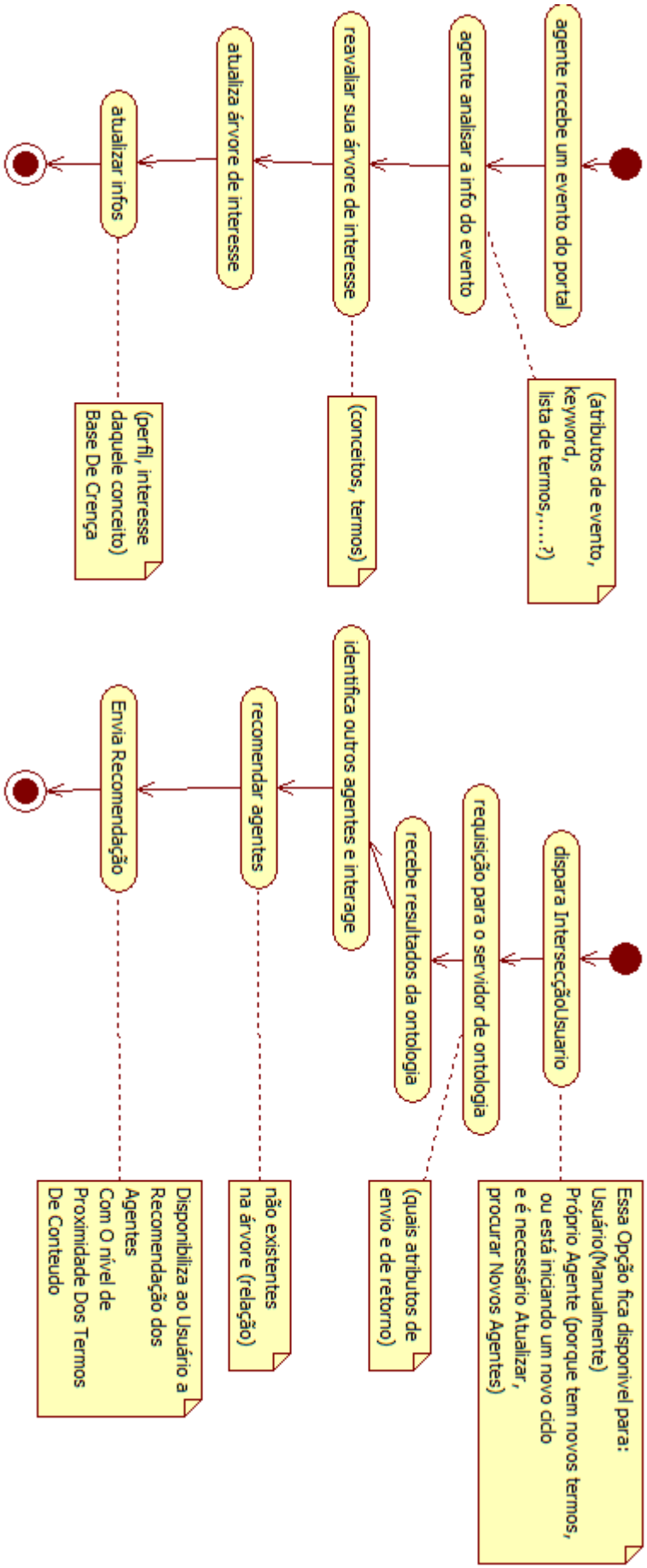

Figura 3.1: Modelo de Execução das Capacidades do Agente do Tipo Usuário

3.2 Especificação

A fase de especificação do sistema na metodologia Prometheus envolve a identificação das metas, cenários, funcionalidades, percepções e papéis.

3.2.1 Funcionalidades

As principais funcionalidades do SMA proposto são:

- *Manutenção do Perfil do Agente Usuário*: consiste em validar as informações do usuário e armazenar na base os dados na base de crença.
- *Recomendação de Agentes*: consiste na comunicação entre os agentes para encontrar intersecções, similaridade de conhecimento baseado em interesses comuns entre os agentes do tipo Usuário.
- *Manutenção do Perfil do Agente Comunidade*: consiste no gerenciamento do perfil da comunidade que é composto pelos interesses da comunidade e pelo resumo do perfil dos usuários "ligados" a esta comunidade.
- Procurar no grupo ao qual a comunidade pertence, usuários que não fazem parte e que poderão fazer parte da comunidade, intersecção definida pelos interesses da comunidade e pelos usuários.

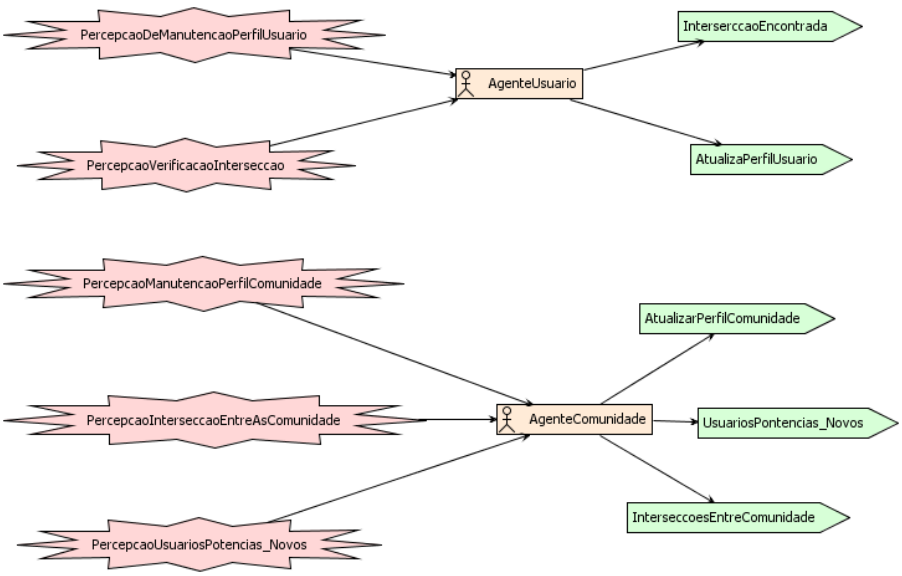

Figura 3.2: Analysis overview - Análise

As funcionalidades estão representadas na Figura 3.2. O *AgenteUsuario* tem como funcionalidade a manutenção de dados do Perfil, esses dados são estáticos (campos pré-definidos) e dinânimos(conhecimento expresso através das interações

na comunidade), e a funcionalidade para a verificação de intersecção entre agentes. Onde o *Agente Origem*, envia mensagem a todos os usuários que pertencem as mesmas comunidades, solicitando aos mesmos, se possuem similaridades de conhecimento. Para cada retorno, é utilizado o Algoritmo 1 para determinar o escore das informações entre os mesmos.

Algorithm 1: Cálculo de intersecção entre agentes usuários

Entrada: Objeto $DadosPerfilOrigem$ com os dados do Agente Origem e $DadosPerfilRetorno$ com os dados do Agente que retornou a solicitação.

Saída: Objeto *Retorno* com o resultado do cálculo de intersecção, valor com a relação do quanto é similar um determinado conhecimento entre os usuários.

```
1  início
2      Score ← 0;
3      para cada Registro ∈ DadosPerfilOrigem faça
4          para cada RegistroRetorno ∈ DadosPerfilRetorno faça
5              se Registro.IdentOntologia = RegistroRetorno.IdentOntologia então
6                  ScoreAux ← 0;
7                  para cada Valor ∈ Registro.Valores faça
8                      para cada ValorRetorno ∈ RegistroRetorno.Valores faça
9                          se Valor.DescricaoValor = ValorRetorno.DescricaoValor
                           então
10                             ScoreAux ← ScoreAux+ValorRetorno.ValorConhecimento;
11                 Score ← Score + (ScoreAux ∗ Registro.PesoOntologia);
12     Retorno ← Score;
```

O Algoritmo 1 cálcula a relação de intersecção entre as usuários, onde:

- *DadosPerfilOrigem* representa os dados de conhecimento do usuário origem;
- *Registro* contém os valores de um conhecimento;
- *DadosPerfilRetorno* representa os dados de conhecimento do usuário de retorno;
- *RegistroRetorno* contém os valores de um conhecimento;
- *Registro.IdentOntologia* é um identificador único de um conhecimento para que o mesmo seja único;
- *Valor* contém os dados de um conhecimento;
- *Registro.Valores* lista de valores de um conhecimento;
- *ValorRetorno.ValorConhecimento* é o quanto um agente conhece em um determinado conhecimento. A classificação e definição desse valor é de responsabilidade do portal;

- *Registro.PesoOntologia* representa o valor do quanto o conhecimento tem importância na estrutura de classificação dos dados, que é informado ao agente pelo portal.

O *AgenteComunidade* é o representante do administrador da comunidade, tem a funcionalidade de manutenção do perfil da comunidade, a localização de intersecção entre os interesses das comunidades e a localização de novos usuários com potencial para a prosperação da comunidade.

Cada funcionalidade tem metas/objetivos que precisam ser executados. Essas metas estão representadas na Figura 3.3, como exemplo a meta *PerfilUsuario* possui submetas. Elas são representadas pelo mesmo componente. Para diferenciá-las é necessário identificar os objetivos conforme sua posição, uma submeta está ligada em uma meta por um traço e o objetivo representado no nível superior é a meta. No Exemplo, são submetas os componentes: *ConsultaDadosUsuario*, *ValidaNovoDadoInOntologiaUsuario* e *AvaliaConflitoUsuario*.

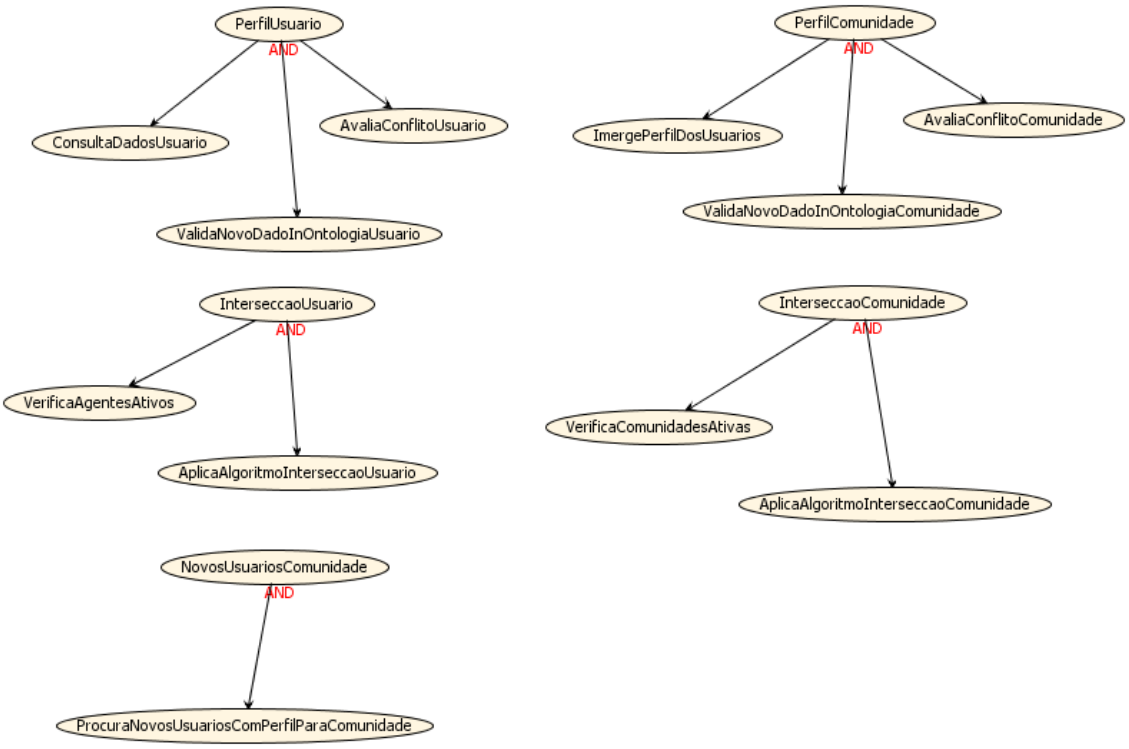

Figura 3.3: Goal Overview - Análise de Metas

3.2.2 Cenários

No SMA analisado, os principais cenários estão representados na Figura 3.4. Cada cenário corresponde a um objetivo e após o cenário ser disparado é executado uma sequência de passos, sendo produzido a saída correta para os dados recebidos.

Os cenários de manutenção de perfil de usuário e comunidade tem por objetivo atualizar na base de crença, dados do perfil dinâmico e estático do agente, e os cenários de intersecção tem por objetivo solucionar o principal objetivo do sistema que é a recomendação de agentes/usuários de acordo com o perfil e conhecimento de cada um. Os principais cenários envolvidos são:

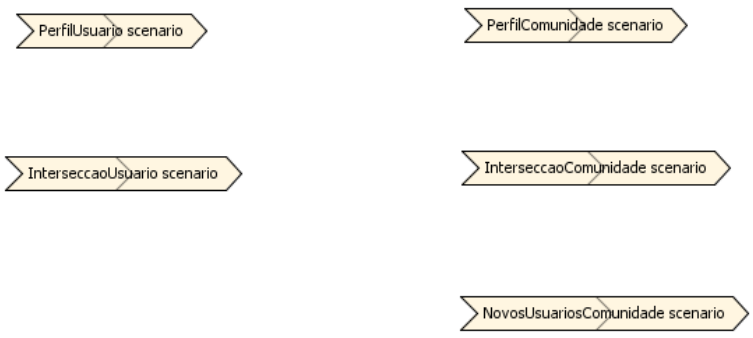

Figura 3.4: Scenarios - Cenários do Sistema SMA

- *PerfilUsuario scenario* é o cenário de manutenção do perfil do usuário que é disparado para solicitar a atualização dos dados de perfil e dados de interesse do *Agente Usuário*. Esse cenário é composto pelas metas (*Consultar Dados Atuais*, *Validar Novo Dado* e *Avalia Conflito da Informação*) e da ação (*Atualização do Perfil do Usuário*).

- *InterseccaoUsuario scenario* é o cenário disparado pelo usuário quando precisa encontrar outros agentes/usuários com compatibilidade de conhecimento e que possa ser útil para a solução de problemas, e também para compartilhar conhecimento. Esse cenário poderá ser executado automaticamente pelo agente em ciclos programados para atualizar e recomendar outros usuários. A execução consiste em enviar mensagens para os outros agentes do mesmo grupo de comunidades, com uma solicitação sobre o conhecimento para o qual necessita ajuda, compartilhamento ou mesmo interação de informação. E a partir dessa requisição todo o retorno é avaliado com base no Algoritmo 1 que calcula o grau de similaridade da informação entre os agentes, e a lista de recomendação com os resultados satisfatórios serão mostrados ao usuário.

- *PerfilComunidade scenario* é o cenário de manutenção do perfil da comunidade que é disparado pelo administrador da comunidade quando solicita a atualização do perfil com as características e interesses da comunidade.

- *InterseccaoComunidade scenario* é o cenário de intersecção entre comunidades que é disparado pelo administrador da comunidade quando o mesmo necessita encontrar outras comunidades com compatibilidade entre seus interesses

ou também pelo próprio agente da comunidade que após um período pode procurar intersecções e apresentar o resultado. Este cenário visa interargir no ambiente e encontrar comunidades relacionadas com interesses comuns. O cálculo de intersecção entre comunidades é realizado pelo Algoritmo 2.

- *NovosUsuariosComunidade scenario* é o cenário que visa encontrar outros usuários com perfil similar a comunidade e ainda não participa. Esse cenário é executado pelo próprio agente em ciclos programados e pelo administrador da comunidade. Após a execução do cenário o resultado com a lista de usuários estará disponível para o administrador. O cálculo para sugestão de novos usuários é realizado pelo Algoritmo 3.

Algorithm 2: Cálculo para Intersecção entre Comunidade

Entrada: Objeto *InteresseComunidadeOrigem* com os dados da Comunidade Origem e *InteresseComunidadeRetorno* com os dados do Agente Comunidade que retornou a solicitação.

Saída: Objeto *Retorno* com o resultado do cálculo score de recomendação, esse valor identifica as comunidades com interesses similares ou parecidos.

```
1  início
2      Score ← 0;
3      para cada Interesse ∈ InteresseComunidadeOrigem faça
4          para cada InteresseRetorno ∈ InteresseComunidadeRetorno faça
5              se Interesse.IdentInteresse = Interesse.IdentInteresse então
6                  ScoreAux ← 0;
7                  para cada Valor ∈ Interesse.ValoresInteresse faça
8                      para cada ValorRetorno ∈ InteresseRetorno.ValoresInteresse
                       faça
9                          se Valor.DescricaoValor = ValorRetorno.DescricaoValor
                           então
10                             ScoreAux ← ScoreAux + ValorRetorno.Qualificador;
11                 Score ← Score + (ScoreAux ∗ Interesse.PesoInteresse);
12     Retorno ← Score;
```

O Algoritmo 2 cálcula a relação de intersecção entre as comunidades, onde:

- *InteresseComunidadeOrigem* representa a lista de interesses da comunidade origem;
- *Interesse* contém os valores de um interesse;
- *Interesse.IdentInteresse* é um identificador único de um interesse para que o mesmo seja único;

- *InteresseComunidadeRetorno* representa a lista de interesses da comunidade destino;
- *InteresseRetorno* contém os valores de um interesse;
- *Interesse.ValoresInteresse* lista de valores de um interesse da comunidade origem;
- *InteresseRetorno.ValoresInteresse* lista de valores de um interesse da comunidade de retorno;
- *Interesse.PesoInteresse* é o quanto um valor é importante da estrutura de ontologia, o mesmo é definido e informado pelo portal;
- *ValorRetorno.Qualificador* representa o valor do quanto a comunidade tem interesse em um conteúdo, a utilização e o valor do mesmo é definido pelo portal e informado ao agente.

Algorithm 3: Cálculo para Recomendação de Novos Usuários

Entrada: Objeto $InteresseComunidadeOrigem$ com os dados da Comunidade Origem e $DadosPerfilRetorno$ com os dados do Agente Usuário que retornou a solicitação.

Saída: Objeto *Retorno* é o resultado do cálculo, esse score de recomendação é o valor do quanto um usuário tem perfil compatível com uma comunidade.

```
1  início
2      Score ← 0;
3      para cada Interesse ∈ InteresseComunidadeOrigem faça
4          para cada RegistroRetorno ∈ DadosPerfilRetorno faça
5              se Interesse.IdentInteresse = RegistroRetorno.IdentInteresse então
6                  ScoreAux ← 0;
7                  para cada Valor ∈ Interesse.ValoresInteresse faça
8                      para cada ValorRetorno ∈ RegistroRetorno.ValoresInteresse faça
9                          se Valor.DescricaoValor = ValorRetorno.DescricaoValor
                           então
10                             ScoreAux ← ScoreAux + ValorRetorno.Qualificador;
11                 Score ← Score + (ScoreAux ∗ Interesse.PesoInteresse);
12     Retorno ← Score/DadosPerfilRetorno.GrauConhecimento;
```

O Algoritmo 3 cálcula o interesse que uma comunidade pode ter em relação a um novo usuário, onde:

- *InteresseComunidadeOrigem* representa a lista de interesses da comunidade;

- *Interesse* contém os valores de um interesse;
- *DadosPerfilRetorno* representa a lista de interesses do usuário que retornou uma solicitação;
- *RegistroRetorno* contém o valor de um interesse;
- *RegistroRetorno.ValoresInteresse* lista de valores de um interesse;
- *Interesse.PesoInteresse* é o quanto um valor é importante da estrutura de ontologia, o mesmo é definido e informado pelo portal;
- *ValorRetorno.Qualificador* representa o valor do quanto o usuário tem interesse em um conteúdo, a utilização e o valor do mesmo é definido pelo portal e informado ao agente.
- *DadosPerfilRetorno.GrauConhecimento* é o valor do quanto o usuário já escreveu ou conhece sobre o interesse. Esse valor é informado ao portal para os agentes, sendo que o agente não define como e de que forma esse valor é qualificado.

3.2.3 Papéis

Os Papéis tem a representação da funcionalidade e no SMA proposto a Figura 3.5 ilustra os papéis que cada Agente participa. O processo de execução de um papel é *Top-Down* ou seja, a ordem de execução é de cima para baixo, com isso, o processo começa com a execução da percepção, após isso serão executadas as metas e o retorno será a ação. Os papéis que estão modelados no Sistema são:

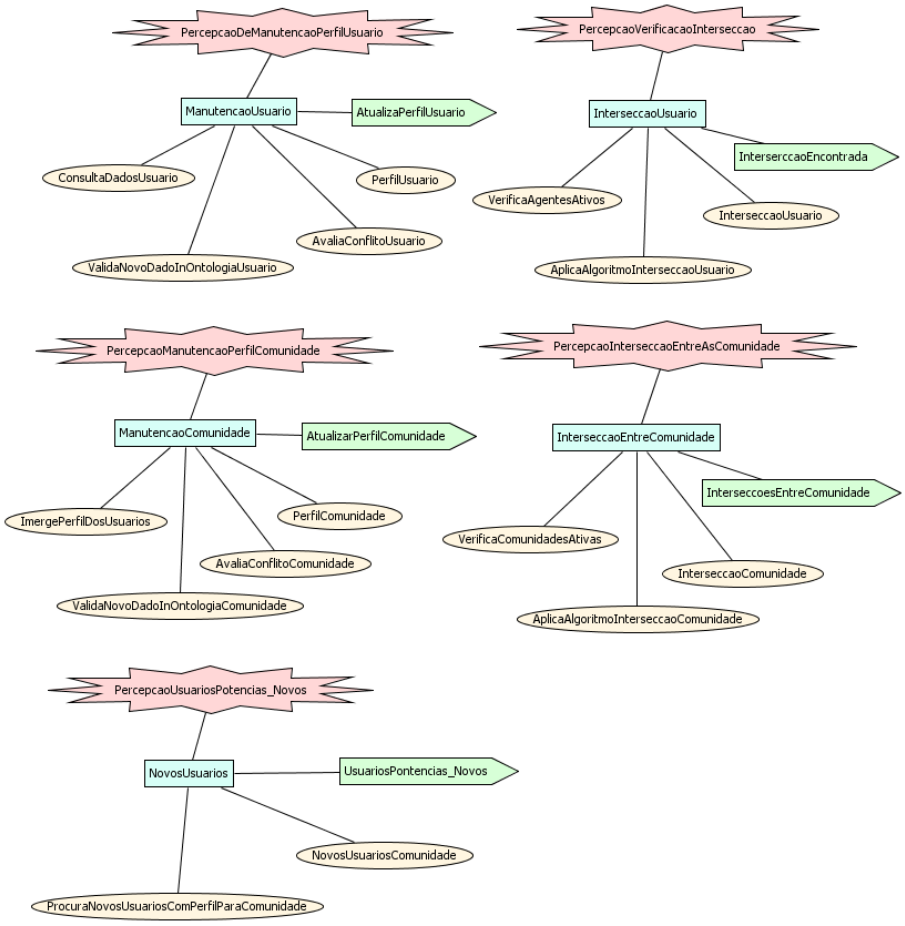

Figura 3.5: System Roles - Papéis

- *ManutencaoUsuario*, funcionalidade responsável por atualizar os dados do usuário na base de crença.Verifica os dados e armazena as informações.
- A *InterseccaoUsuario* é a funcionalidade que procura por intersecção com outros agentes. Com base nas crenças, informações dinâmicas, com referência em uma ontologia, são calculados o escore de similaridade entre os agentes que respondem as solicitações e depois recomendados ao usuário origem.
- *ManutencaoComunidade*, consiste em atualizar a base do perfil da comunidade, com os interesses e características da comunidade e o resumo das informações dos agentes que compõem a comunidade.

- A *InterseccaoEntreComunidade* tem como objetivo encontrar outras comunidades que tenham interesses, intersecções, populando a rede de relacionamento da comunidade e aumentando a possibilidade de novos usuários para qualquer uma das comundidades da base de relacionamento.

- O objetivo do papel *NovosUsuarios* é encontrar novos usuários para a comunidade. Usuários com interesses em comum com a comunidade e que estão em alguma comunidade relacionada a comunidade que está executando à funcionalidade de encontrar novos usuários. Neste papel o agente comunidade executa a percepção *PercepcaoUsuariosPotencias_Novos* e a partir disso são localizados agentes das comunidades que estão interligadas ao agente e sugeridos ao administrador da Comunidade. Assim é possível encontrar novos usuários com perfil da comunidade.

3.2.4 Percepções

A comunicação do ambiente com os agentes, os eventos que o agente pode diagnosticar, receber e responder, na metodologia Prometheus é modelado através de *Percepções* e as que o sistema proposto possui são:

- *PercepcaoDeManutencaoPerfilUsuario* recebe os dados de atualização do perfil do usuário.

- *PercepcaoVerificacaoInterseccao* é executada quando é solicitado para que um agente se comunique com os outros a procura de similaridades de conhecimento e quando o agente recebe uma mensagem de outro solicitando uma similaridade de conhecimento e esse retorna os dados que possui no mesmo nível da ontologia.

- *PercepcaoManutencaoPerfilComunidade* recebe os dados e atualiza o perfil da comunidade.

- *PercepcaoInterseccaoEntreAsComunidade* é executada quando é solicitado para que o agente comunidade localize outras comunidades com similaridade nos interesses da comunidade e no momento que o mesmo precisa responder a mensagens de outros agentes comunidade.

- *PercepcaoUsuariosPotencias_Novos* é executada quando recebe uma solicitação para procurar novos usuários para a comunidade, usuários com perfil compatível com a comunidade.

3.2.5 Definição da Arquitetura

Na definição da arquitetura, temos a modelagem de dados que cada agente acessa (Figura 3.6), o relacionamento dos agentes (Figura 3.7), o agrupamento dos papéis por agente(Figura 3.8) e a relação dos agentes com os dados.

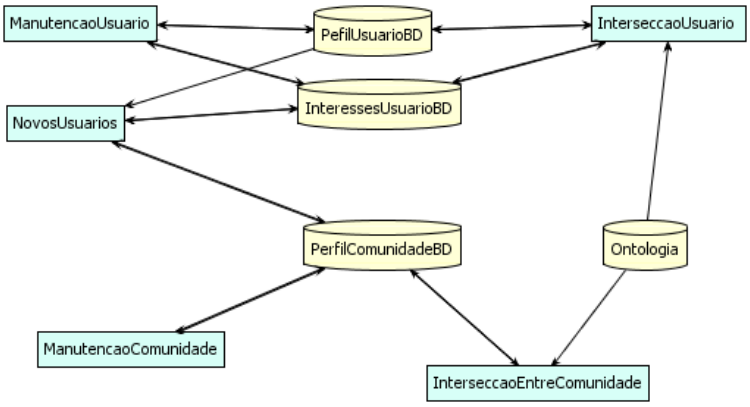

Figura 3.6: Data Coupling - Modelagem de Dados

Figura 3.7: Agent Acquaintance - Relacionamento entre Agentes

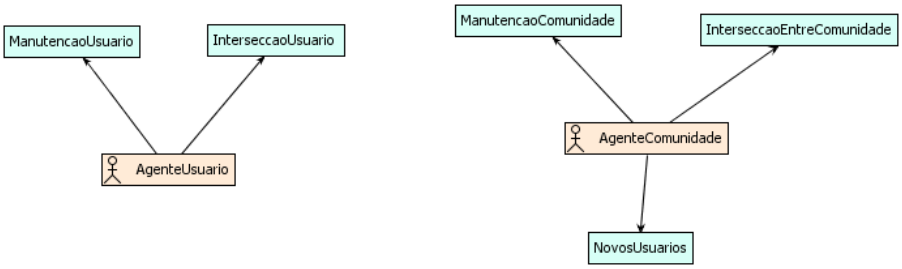

Figura 3.8: Agent Role Grouping - Grupo de Papéis por Agente

A Figura 3.9 ilustra a modelagem dos protocolos de comunicação de um agente com outro e a consulta ou gravação que cada agente faz na base de dados.

3.2.6 Agentes

No SMA proposto foi necessário utilizar o agente *AgenteUsuario*, responsável pelas funcionalidades do usuário e este será instanciado por cada usuário no ambiente e o agente *AgenteComunidade* com as funcionalidades da comunidade, onde cada uma tem seu próprio agente executando e fornecendo as funcionalidades de manutenção, intersecção e localização de novos usuários.

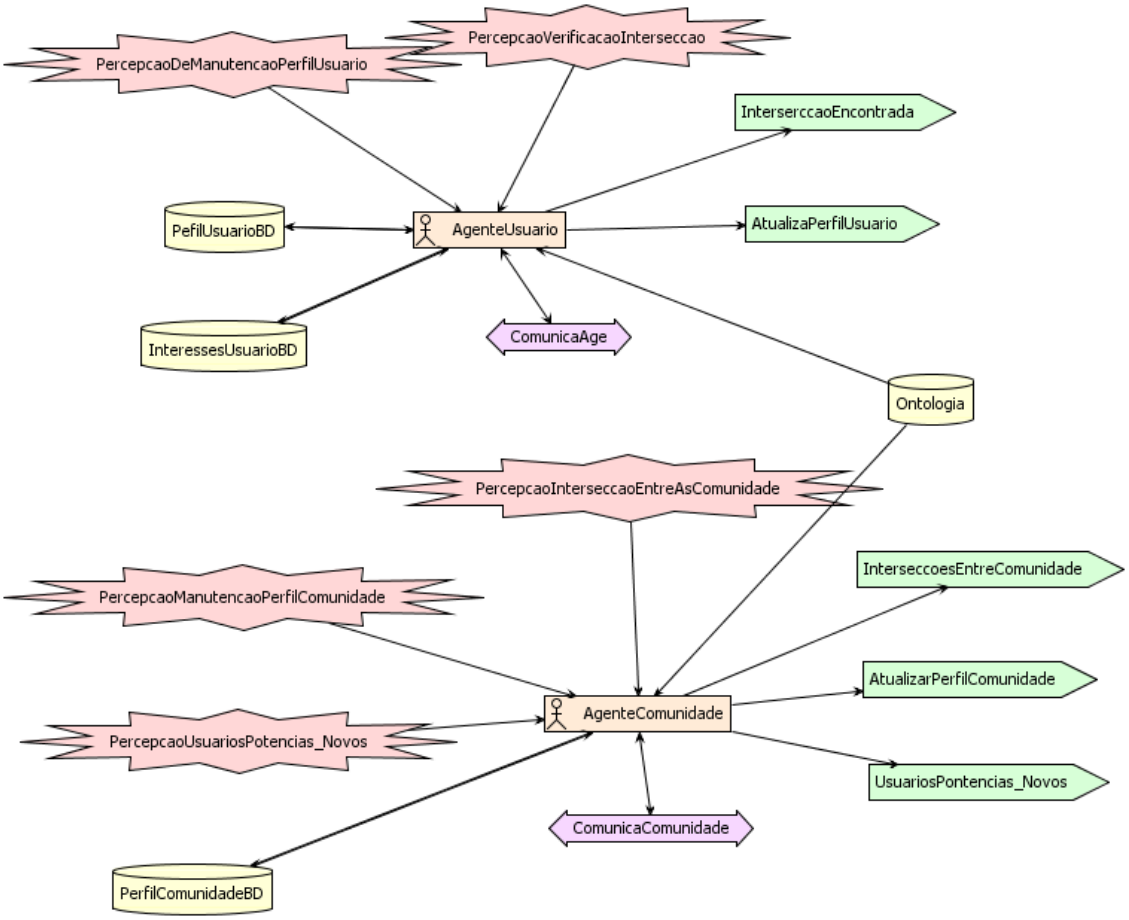

Figura 3.9: System Overview - Sistema

3.2.7 Detalhamento do Projeto

Na fase de detalhamento de projeto são especificados as capacidades e planos de cada agente. Na sequência será apresentado os agentes do Sistema, capacidades e planos de cada um.

3.2.7.1 Agente Usuário

Esse agente é responsável pela manutenção do perfil do usuário e por disponibilizar as recomendações de outros agentes com conhecimentos relacionados. A Figura 3.10 mostra as percepções, ações, mensagens e base de dados relacionadas com suas capacidades.

As capacidades do agente usuário são: *Capacidade de Manutenção de Dados do Usuário* (Figura 3.11 e Algoritmo 4) e *Capacidade de Intersecção entre Usuários* (Figura 3.12 e Algoritmo 5).

O Algoritmo 4 atualiza na base de crenças do usuário as alterações de preferências de conhecimentos adquiridos ou expostos nas comunidades, onde:

- *DadosAtualizacaoPerfil* representa a lista de dados com as atualizações;

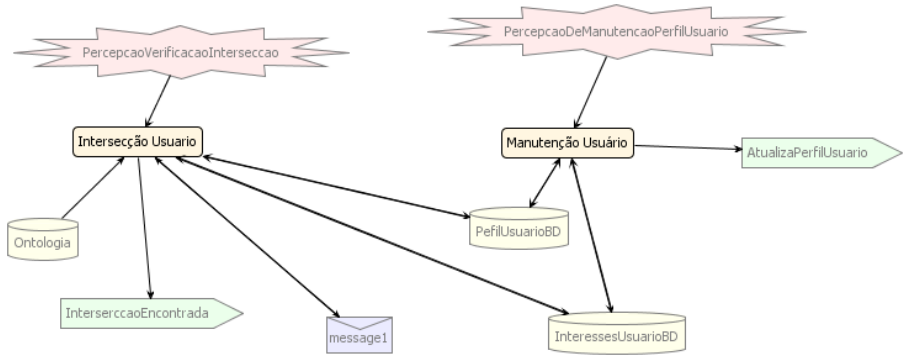

Figura 3.10: Agente Usuário

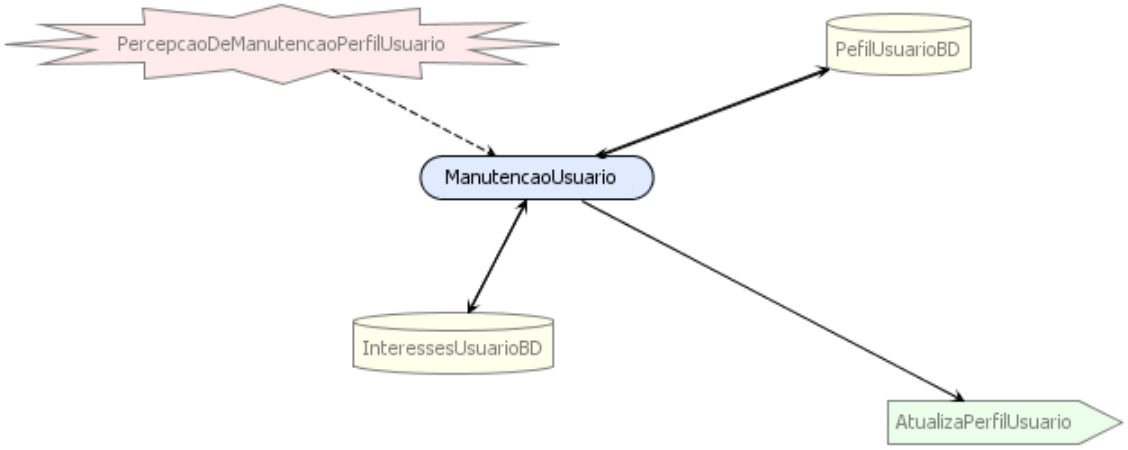

Figura 3.11: Capacidade de Manutenção de dados do Usuário

- *Dado* contém os valores de interesse e referência do dado na Ontologia;
- *CheckBase(Dado)* verifica na base de conhecimento do usuário se esse (*Dado*) já existe;
- *DadoInBase* contém o (*Dado*) encontrado na base e caso contrário um objeto (*vazio*);
- *AtualizaDadosNaBase(DadoInBase, Dado)* é o algoritmo que faz a atualização do *Dado* encontrado e atualiza o valor de escore desse conhecimento;
- *AdicionaDadosNaBase(Dado)* é o algoritmo que adiciona o novo *Dado* na base de conhecimento do usuário;
- *Retorno* ← *StatusOperacao* : atribuição do resultado da atualização ao objeto de retorno.

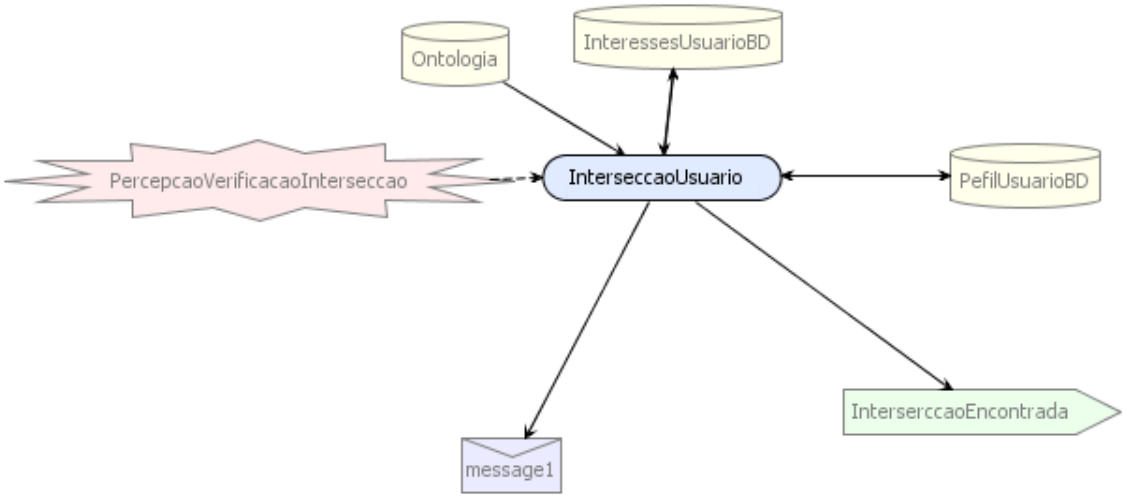

Figura 3.12: Capacidade de Intersecção entre Usuários

Algorithm 4: ManutencaoUsuario

Entrada: Objeto $DadosAtualizacaoPerfil$ com os dados para a atualização do Perfil do Usuário.

Saída: Objeto *Retorno* com os dados do resultado da atualização.

```
1 início
2     Recebe do Usuário uma solicitação para atualizar informações na base de conhecimento;
3     para cada Dado ∈ DadosAtualizacaoPerfil faça
4         DadoInBase ← CheckBase(Dado);
5         se DadoInBase <> vazio então     /* Verifica se encontrou o Registro */
6             AtualizaDadosNaBase(DadoInBase, Dado);
7         senão
8             AdicionaDadosNaBase(Dado);
9         Retorno ← StatusOperacao;
```

O Algoritmo 5 tem por objetivo retornar uma lista de possíveis usuários com informações relacionadas ao $Agente_O$. Essa relação, de similaridade de conhecimento é verificada com todos os usuários que participam das mesmas comunidades do $Agente_O$. Essa solicitação poder ser realizada pelo usuário ou automaticamente pelo seu agente em ciclos pré-definidos. A informação com a lista de usuários é disponibilizada para o usuário e para a comunidade através da Interface.

- *PerfilUsuarioOrigem* contém as informações que servirão para realizar a consulta e base para a verificação de perfis compatíveis.
- *registro* contém os dados de cada informação que necessita ser verificada.
- $AG(Interest_i)$ representa a lista do $Agente_O$ contendo os usuários j com interesses similares;

- $Interest_i$ contém o termo representando o interesse analisado do perfil do agente O;
- *CheckOntologicalBase(registro)* esta função solicita ao programa que gerencia a organização de termos do ambiente, a análise do termo/valor(*registro*);
- $sendUserAgent_j(Interest_i)$ envia para o Agente j as informações retornadas pela Ontologia, para os outros agentes Usuário avaliarem em sua base de crenças a compatibilidade com seus próprios interesses;
- $Agent_j$ é o Agente que está disponível para receber a mensagem de envio.
- *ComunidadesOrigem* é o objeto que representa todas as comunidades da qual o agente O pertence.
- $Interest_j \leftarrow receiveUserAgents()$ recebe do agente j os interesses relacionados;
- $Score_j$ armazena o grau de interesse do agente j sobre o termo $Interest_i$;
- *CalculateInterestScore()* é o Algoritmo 1 que calcula o grau de interesse com o agente j, levando em consideração somente as informações dinâmicas, ou seja, conhecimentos que são criados pelo usuário/agente O ao participar da comunidade.
- $Retorno \leftarrow AG(Interest)$: mensagem de retorno recebe a lista de agentes recomendados para a informação válida.

Algorithm 5: InterseccaoUsuario

Entrada: Objeto $PerfilUsuarioOrigem$ com os termos de conhecimento do usuário.

Saída: Mensagem *Retorno* com a lista de usuários recomendados para o usuário origem.

```
1  início
2      O agente origem (Agente_O) recebe do Usuário ou automaticamente após cada ciclo
       uma solicitação para buscar agentes/usuários com conhecimentos similares ou iguais
       aos termos de conhecimento do usuário que faz a solicitação;
3      para cada registro ∈ PerfilUsuarioOrigem faça
4          Interest_i ← CheckOntologicalBase(registro);
5          para cada Agent_j ∈ ComunidadesOrigem faça
6              sendUserAgent_j(Interest_i);
7          para cada Agent_j ∈ ComunidadesOrigem faça
8              Interest_j ← receiveUserAgents();
9              Score_j ← CalculateInterestScore();
10             AG(Interest_i) ← AG(Interest_i) ∪ AG(Interest_j) + Score_j;
11     Retorno ← AG(Interest);
```

3.2.7.2 Agente Comunidade

Esse agente tem a responsabilidade de consultar recomendações de outras comunidades, disponibilizando-as ao administrador. Gerenciar as informações do perfil e encontrar novos usuários, com perfil compatível àquela comunidade entre a lista de comunidades relacionadas. A Figura 3.13 mostra as percepções, ações, mensagens e base de dados relacionadas com suas capacidades. As capacidades são: *Capacidade de Manutenção da Comunidade* (Figura 3.14 e Algoritmo 6), *Capacidade de Intersecção Entre Comunidades*(Figura 3.15 e Algoritmo 7) e *Capacidade de Encontrar Novos Usuários*(Figura 3.16 e Algoritmo 8).

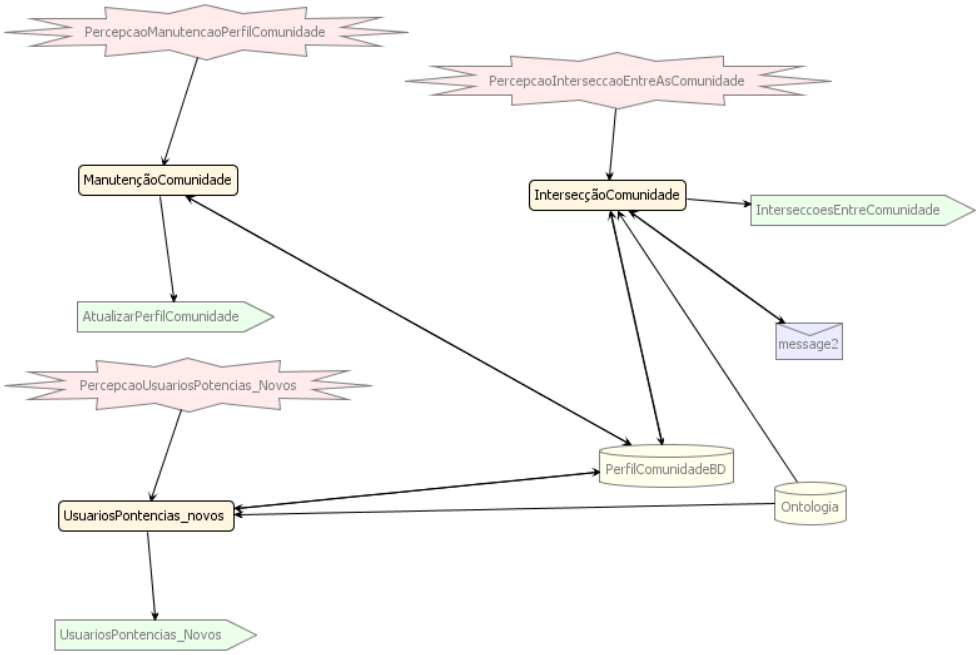

Figura 3.13: Agente Comunidade

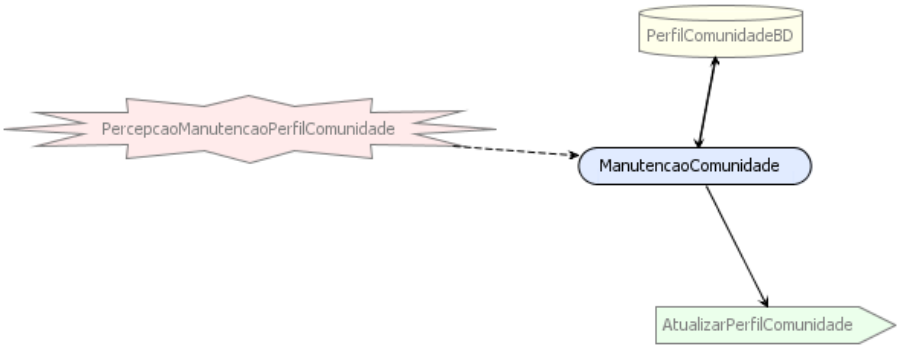

Figura 3.14: Capacidade de Manutenção da Comunidade

Algorithm 6: ManutencaoComunidade

Entrada: Objeto $DadosAtualizacaoPerfil$ com os dados para a atualização do Perfil da Comunidade.

Saída: Objeto *Retorno* com os dados de status da atualização.

```
1 início
2     Recebe do Administrador da Comunidade uma solicitação para atualizar informações
      na base de conhecimento;
3     para cada Dado ∈ DadosAtualizacaoPerfil faça
4         DadoInBase ← CheckBase(Dado);
5         se DadoInBase <> vazio então     /* Verifica se encontrou o Registro */
6             AtualizaDadosNaBase(DadoInBase, Dado);
7         senão
8             AdicionaDadosNaBase(Dado);
9         Retorno ← StatusOperacao;
```

O Algoritmo 6 atualiza na base de crenças do administrador da comunidade as alterações de preferências, o objetivo desse plano é atualizar na base de crenças da comunidade com as alterações de perfil da comunidade. Além de imergir o perfil dos usuários desta comunidade para ter em sua base um resumo do perfil dos usuários conectados a ela, onde:

DadosAtualizacaoPerfil representa a lista de dados com as atualizações;

Dado contém os valores de interesse e referência do dado na Ontologia;

CheckBase(Dado) verifica na base de conhecimento da conhecimento se esse (*Dado*) já existe;

DadoInBase contém o (*Dado*) encontrado na base e caso contrário um objeto (*vazio*);

AtualizaDadosNaBase(DadoInBase, Dado) é o algoritmo que para o *Dado* encontrado faz a atualização da informação;

AdicionaDadosNaBase(Dado) é o algoritmo que adiciona o novo *Dado* na base de conhecimento da comunidade;

Retorno ← *StatusOperacao* : atribuição do resultado da atualização.

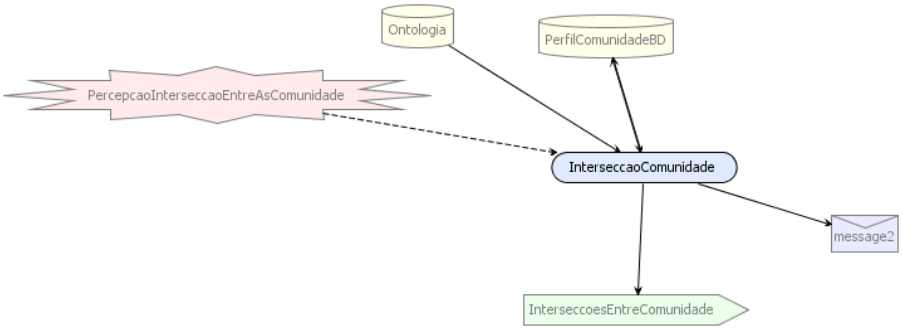

Figura 3.15: Capacidade de Intersecção Entre Comunidades

Algorithm 7: InterseccaoComunidade

Entrada: Objeto $PerfilComunidadeOrigem$ com as características da comunidade.

Saída: Mensagem *Retorno* com a lista de comunidades recomendadas para a comunidade.

1 **início**
2 O agente origem ($Agente_O$) recebe do administrador ou automaticamente após cada ciclo uma solicitação para buscar agentes/comunidade com conhecimentos similares ou iguais as características da que faz a solicitação;
3 **para cada** $registro \in PerfilComunidadeOrigem$ **faça**
4 $Interest_i \leftarrow CheckOntologicalBase(registro)$;
5 **para cada** $Agent_j \in Comunidades$ **faça**
6 $sendComAgent_j(Interest_i)$;
7 **para cada** $Agent_j \in PerfilComunidadeOrigem$ **faça**
8 $Interest_j \leftarrow receiveComAgent()$;
9 $Score_j \leftarrow CalculateComunInterestScore()$;
10 $AG(Interest_i) \leftarrow AG(Interest_i) \cup AG(Interest_j) + Score_j$;
11 $Retorno \leftarrow AG(Interest)$;

O Algoritmo 7 tem por objetivo retornar uma lista de possíveis comunidades com informações relacionadas ao $Agente_O$. A relação, de similaridade de conhecimento é verificada em todas as comunidades. Essa solicitação pode ser realizada pelo administrador ou automaticamente pelo seu agente em ciclos pré-definidos. A informação com a lista de comunidades é disponibilizada para o administrador através da Interface.

PerfilComunidadeOrigem contém as informações que servirão para realizar a consulta e base para a verificação de perfis compatíveis.

registro contém os dados de cada informação que necessita ser verificada.

$AG(Interest_i)$ representa a lista do $Agente_O$ contendo as comunidades *j* com interesses similares;

$Interest_i$ contém o termo representando o interesse analisado do perfil do agente *O*;

CheckOntologicalBase(registro) solicita a análise desse termo/valor(*registro*) para a base de Ontologia. Essa base é definida e controlada pelo portal;

$sendComAgent_j(Interest_i)$ envia para o Agente *j* as informações retornadas pela Ontologia, para os outros agentes comunidade avaliarem em sua base de crenças a compatibilidade com seus próprios interesses;

$Agent_j$ é o Agente que está disponível para receber a mensagem de envio.

Comunidades é o objeto que representa todas as comunidades da qual o agente O pertence.

$Interest_j \leftarrow receiveComAgent()$ recebe do agente j os interesses relacionados;

$Score_j$ armazena o grau de interesse do agente j sobre o termo $Interest_i$;

CalculateComunInterestScore() é o Algoritmo 2 que calcula o grau de interesse com o agente j, levando em consideração somente as informações estáticas, ou seja, preferências do perfil do agente$O$ ao definir a comunidade.

$Retorno \leftarrow AG(Interest)$: mensagem de retorno recebe a lista de agentes recomendados para a informação válida.

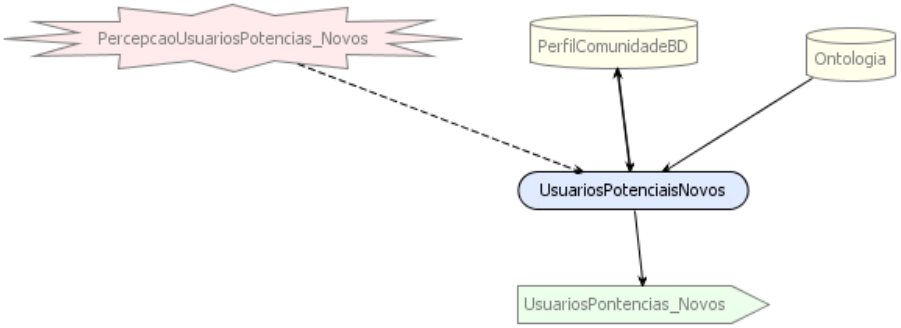

Figura 3.16: Capacidade de Encontrar Novos Usuários

Algorithm 8: UsuariosPotenciaisNovos

Entrada: Objeto $PerfilComunidadeOrigem$ com os dados do perfil da comunidade.

Saída: Objeto *Retorno* com a lista de usuários com perfil compatível com a da Comunidade.

```
1 início
2     Recebe do Agente Comunidade ou automaticamente após cada ciclo uma solicitação
      para buscar novos agentes/usuários com intersecção com o perfil da comunidade ;
3     para cada registro ∈ PerfilComunidadeOrigem faça
4         Interest_i ← CheckOntologicalBase(registro);
5         sendAllUserAgents(Interest_i);
6         Interest_j ← receiveAllUserAgents();
7         Score_j ← CalculateInterestScoreNewUsers();
8         AG(Interest_i) ← AG(Interest_i) ∪ AG(Interest_j) + Score_j;
```

O Algoritmo 8 realiza uma busca por intersecções de interesses dos usuários que pertencem ao grupo da comunidade que está solicitando. Aqueles com perfil que se assemelha ao perfil da comunidade são recomendados, onde:

AG($Interest_i$*)* representa a lista do agente i contendo os usuários j com interesses similares;

$Interest_i$ contém o termo representando o interesse analisado do perfil do agente i;

CheckOntologicalBase(registro) solicita a análise desse termo/valor(*registro*) para a base de Ontologia;

sendAllUserAgents($Interest_i$*)* com as informações retornadas pela Ontologia, o agente i envia para os outros agentes Usuário avaliarem em sua base de crenças a compatibilidade com seus próprios interesses;

$Interest_j \leftarrow receiveAllUserAgents()$ recebe de cada agente j os interesses relacionados;
$Score_j$ armazena o grau de interesse dos agentes j sobre o termo $Interest_i$;
CalculateInterestScoreNewUsers() é o algoritmo 3 que calcula o grau de interesse com o Agente j.

3.3 Considerações Finais

Este Capítulo apresentou a modelagem do SMA proposto utilizando a metodologia Prometheus. Através das três fases da liguagem de modelagem e da ferramenta gráfica disponível no Prometheus, foi possível obter a modelagem conceitual do SMA que prototipa a solução da problemática contida neste trabalho. Conclui-se que os diagramas da metodologia Prometheus abordam uma visão geral e detalhada do sistema e permitem especificar por completo o comportamento do agente em seus vários contextos. As visões dos agentes, capacidades, planos, cenários e a arquitetura geral fornecem uma modelagem adequada para a representação conceitual de um SMA e permitem a implementação conforme os requisitos estabelecidos. A próxima etapa consiste na implementação da especificação definida neste capítulo. Para tanto, a utilização do *Prometheus Design Tool* (PDT)[1], forneceu a geração dos esqueletos iniciais de código JAVA, destinados à plataforma JACK, que tivemos que adaptar para a plataforma JADE, com algumas modificações.

[1]http://www.cs.rmit.edu.au/agents/pdt/; acessado em set/2010

4 IMPLEMENTAÇÃO

No capítulo anterior foi apresentado a modelagem da solução do Sistema SMA utilizando a metodologia Prometheus. Nesse capítulo será ilustrado as principais classes que compõem a solução, diagramas de classe e diagrama de sequência para auxiliar na explicação. A solução está escrita na linguagem Java e com auxílio de recursos da plataforma Jade. A IDE Eclipse com plugins Jade e o NetBeans foram as ferramentas utilizadas para o desenvolvimento os agentes e as interfaces de protótipo.

4.1 Classes

Classes são objetos com características e funcionalidades similares, composta por métodos e atributos definindo comportamentos e formas de execução para realizar seus objetivos no Sistema. O Sistema é composto por classes que representam agentes, capacidades, percepções, planos, mensagens e armazenamento de dados em uma estrutura organizada. As principais classes do Sistema são: *AgenteUsuario* e *AgenteComunidade*.

4.1.1 AgenteUsuario

Esta classe representa um agente do tipo usuário, herda as características e comportamentos da classe *Agent* da plataforma Jade, sendo necessário sobrescrever os métodos *setup* e *takeDown* para executar regras quando o agente é inicializado e finalizado. Entre as funcionalidade e comportamentos, o principal objetivo é recomendar usuários de acordo com a intersecção de informações e armazenar os dados do perfil do agente. Os principais métodos da classe são:

- *setup*: inicializador do agente, cria a instância das classes atributos e executa a rotina com as chamadas dos métodos de carga do perfil e registro do agente no DF (Directory Facilitator) da plataforma Jade;

- *CarregaDadosPerfil*: busca os dados do perfil e os armazena em memória na

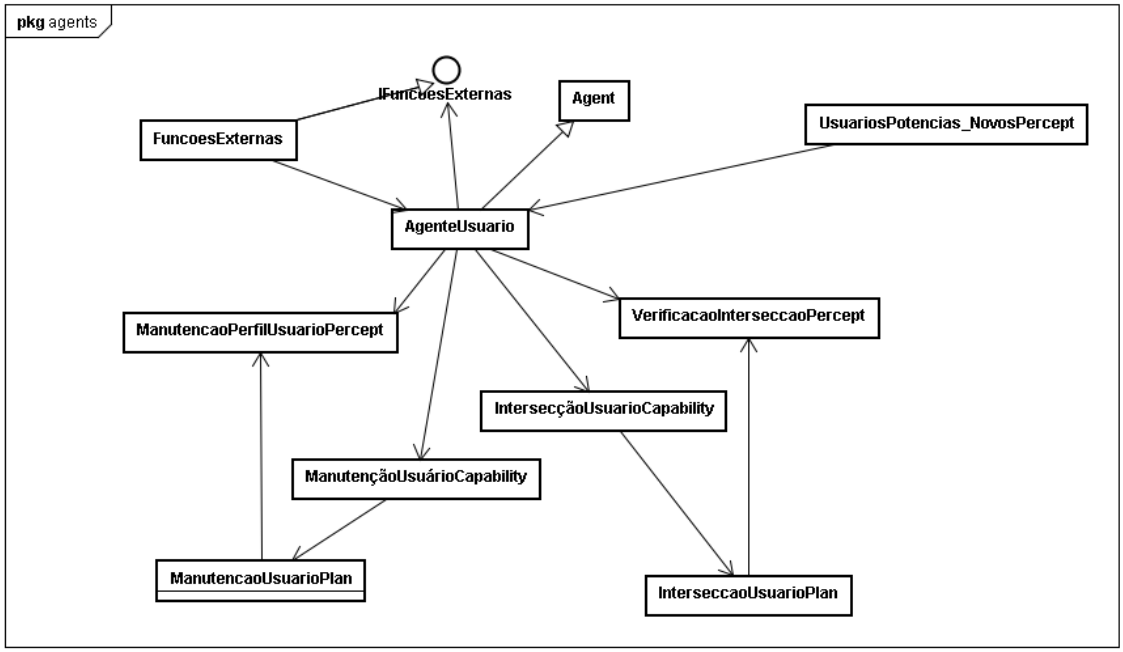

Figura 4.1: Diagrama de Classes - Agente Usuário

estrutura de dados definida pelas classes *ConhecimentoAgente* e *InteressesUsuario*;

- *RegistrarAgente*: é o método que realiza o registro do agente usuário no DF. O registro consiste em setar os serviços do agente(comunidades que participa), e o nome para que o mesmo possa ser localizado e identificado para estar apto à enviar e receber mensagens no ambiente;
- *GravarDadosPerfil*: armazena os dados do perfil do agente em arquivo;
- *DestruirAgente*: solicita a remoção do agente do registro DF;
- *IniciaInterserccaoEncontrada*: método que solicita a execução da capacidade de procurar agentes relacionados aos termos de conhecimento ainda não consultados e recomendados;
- *AdicionarInterseccao*: armazena o grau de intersecção com os usuários que responderam a solicitação;
- *takeDown*: método que é executado quando um agente é finalizado.

As classes atributo e a relação entre elas estão definidas no diagrama de classes da Figura 4.1. Nesta, temos as relações e as classes com as percepções, capacidades e planos que são necessárias para a execução das funcionalidades do agente usuário.

A classe *Agent* da plataforma Jade tem as funcionalidades e métodos necessários para que toda a classe que extende a ela, possa ter os comportamentos de um agente.

Com isso é possível registrar o agente no DF, executar o agente no container Jade e adicionar os comportamentos que podem receber e enviar mensagens.

Ao inicializar o agente é necessário registrar o agente na plataforma Jade, o que é feito pelo código

```
public void RegistrarAgente(Agent agent, String name, String[] serviceType)
{
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.setName(agent.getAID());
    ServiceDescription sd = null;
    for (int i = 0; i < serviceType.length; i++)
    {
        sd = new ServiceDescription();
        sd.setName(name);
        sd.setType(serviceType[i]);
        dfd.addServices(sd);
    }
    sd = new ServiceDescription();
    sd.setName(name);
    sd.setType("AgenteUsuario");
    dfd.addServices(sd);
    try
    {
        DFService.register(agent, dfd);
    }
    catch (FIPAException fe)
    {
        System.out.println("Register exception: " + fe.getMessage());
    }
}
```

e adicionar os comportamentos ao agente que é feito pelo código

```
...
ThreadedBehaviourFactory tbf = new ThreadedBehaviourFactory();
this.addBehaviour(tbf.wrap(eventUsuariosPotencias_NovosPercept));
this.addBehaviour(tbf.wrap(eventVerificacaoInterseccaoPercept));
...
```

4.1.2 AgenteComunidade

A classe *AgenteComunidade* representa o agente do tipo comunidade e herda as características e comportamentos da classe *Agent* da plataforma Jade, sendo que é necessário sobrescrever os métodos *setup* e *takeDown* para executar regras necessárias no momento em que o agente é inicializado e finalizado. O principal objetivo é procurar novos usuários, localizar intersecções entre comunidades e armazenar os dados do perfil de interesses. Os principais métodos da classe são:

- *setup*: inicializador do agente, cria a instância das classes que fazem parte dos atributos da classe e executa a rotina de chamada dos métodos que fazem a carga do perfil da comunidade e o registro no DF (Directory Facilitator) da plataforma Jade;

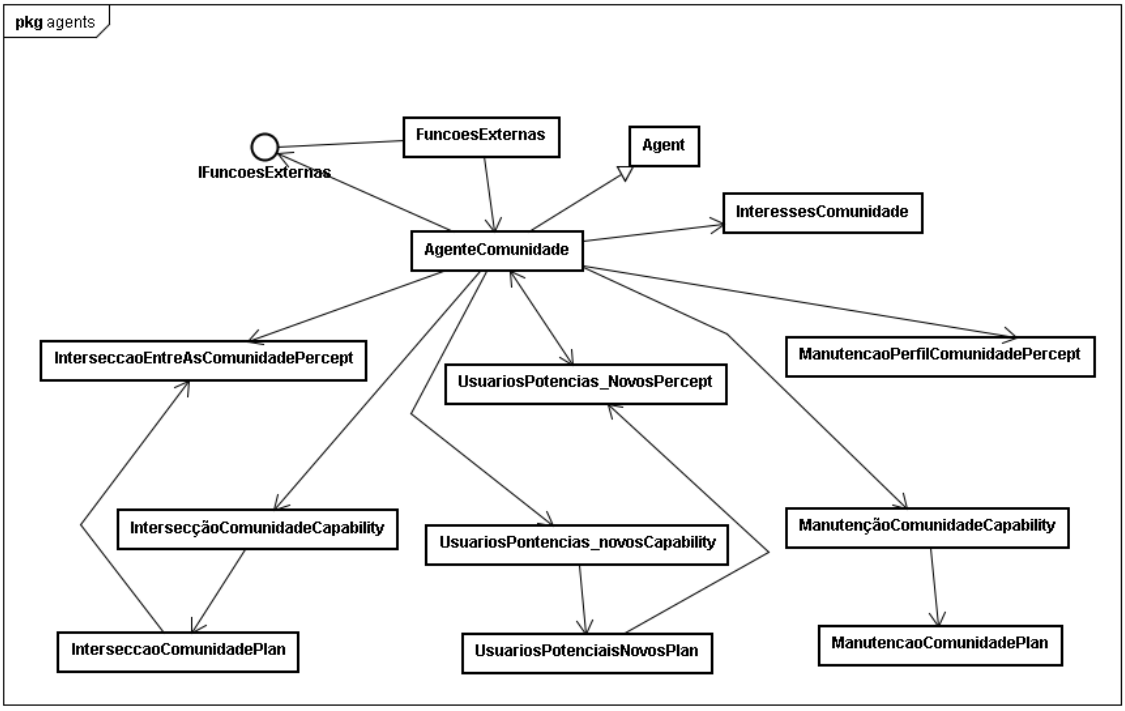

Figura 4.2: Diagrama de Classes - Agente Comunidade

- *RegistrarAgente*: o método realiza o registro do agente no DF. O registro consiste em setar o tipo e nome para ser localizado e identificado, estando apto à enviar e receber mensagens no ambiente;
- *CarregaDadosPerfil*: busca os dados do perfil do agente comunidade e os armazena em memória;
- *GravarDadosPerfil*: armazena os dados do perfil do agente comunidade em arquivo;
- *DestruirAgente*: solicita a remoção do agente do registro DF;
- *IniciaUsuariosPontencias_Novos*: executa a capacidade de procurar novos usuários para a comunidade;
- *IniciaInterserccaoEncontrada*: executa a capacidade de procurar a intersecção entre comunidade;
- *AdicionarNovosUsuarios*: adiciona na lista de recomendação de usuários o usuário com interesse em comum com a comunidade e o resultado do cálculo de interesse da comunidade com o usuário;
- *AdicionarInterseccao*: adiciona na lista de intersecção de comunidades a comunidade que respondeu a solicitação e o resultado de cálculo de interesse com a comunidade.

- *takeDown*: método que é executado quando um agente é finalizado.

A estrutura de classes utilizadas para a modelagem das fucionalidades do agente comunidade estão representadas na Figura 4.2. Nesta, temos as relações e as classes com as percepções, capacidades e planos.

Ao inicializar o agente comunidade é necessário registrar o agente na plataforma Jade, o que é feito pelo código

```
public void RegistrarAgente(Agent agent, String name)
{
    DFAgentDescription dfd = new DFAgentDescription();
    dfd.setName(getAID());
    ServiceDescription sd = new ServiceDescription();
    sd.setType("AgenteComunidade");
    sd.setName(name);
    dfd.addServices(sd);
    try
    {
        DFService.register(this, dfd);
    }
    catch (FIPAException fe)
    {
        fe.printStackTrace();
    }
}
```

4.1.3 Classes de capacidade

Uma capacidade representa uma funcionalidade do agente no Sistema. Cada uma tem um ou mais plano com as regras e lógicas para atingir o objetivo da capacidade. Essas capacidades são representadas pelas classes citadas na sequência.

No agente usuário a classe *ManutençãoUsuário* possui a rotina, métodos e planos necessários para realizar a manutenção dos dados do perfil do usuário(consulta, inclusão e atualização). O método com a funcionalidade de executar o plano de manutenção do perfil dinâmico *AtualizaPerfilUsuario(Conhecimento[] InteressesDinamicos)*. Quando é uma atualização, já existe esse conhecimento no perfil, é alterado o valor com a quantidade qualificadora do quanto o agente possui de conhecimento sobre esse termo. Para a manutenção do perfil estático com os interesses do usuário é necessário o método *AtualizaPerfilUsuario(Interesse[] InteressesStatico)*.

E a classe *IntersecçãoUsuario* tem a rotina, métodos e planos necessários para procurar usuários relacionados ao conhecimento do agente. Os planos necessários para buscar usuários com conhecimento similar é feita pelo método *IniciaInterserccaoEncontrada* e *InterserccaoEncontrada* executa os planos que determinam o cálculo de recomendação do agente que respondeu a solicitação.

As classes com as capacidades do agente comunidade estão representadas pela classe *ManutençãoComunidade* que possui a rotina, métodos e planos necessários

para atualizar os dados do perfil da comunidade. O método *AtualizarPerfilComunidade* executa o plano de manutenção de dados do perfil da comunidade. E a classe responsável pela capacidade de recomendar comunidades com interesses similares é *IntersecçãoComunidade*. Nesta o método de *IniciaInterseccoesEntreComunidade* executa os planos necessários para encontrar novas comunidades e *InterseccoesEntreComunidade* utiliza as informação de similaridade de outros agentes para executar o plano que determina o grau de intersecção entre os agentes.

Para que a comunidade possa evoluir e proporcionar um número amplo de usuários com o conhecimento importante para os interesses da comunidade é necessário encontrar sempre usuários com potencial e perfil compatível. Essa capacidade é representada pela classe *UsuariosPontencias_novos* e os métodos que possui para tornar viável a recomendação de novos usuários é o *IniciaUsuariosPontencias_Novos* com o plano para localizar novos usuários e o *UsuariosPontencias_Novos* que a partir do retorno obtido, calcula o nível de interesse nesse usuário.

4.1.4 Classes para armazenar informação

As classes mencionadas nessa seção, são classes utilizadas para armazenar dados com o mapeamento entre conceito da ontologia e informação em uma estrutura organizada e implementando a interface *Serializable* do Java para que seja possível serializar os dados, assim os dados podem ser salvos em arquivo e os mesmos ser transmitidos como Mensagem entre os agentes. Para acessar o atributo de uma classe é necessário que o mesmo seja feito por métodos get e set, seguindo o conceito de encapsulamento.

O conhecimento do agente usuário é armazenado na classe *ConhecimentoAgente* que contém uma lista de objetos do tipo *Conhecimento* disponível pelo atributo *ArrayList<Conhecimento> listaConhecimentos*. Essa lista é utilizada para obter o cálculo similaridade entre os usuários.

O tipo *Conhecimento* é a classe que identifica a informação com um atributo *IdOntologia* que representa o tipo de conhecimento, identificador na ontologia para assim as informações terem equivalência e ser a respeito do mesmo assunto. Para cada conhecimento o atributo *PesoNaOntologia* é a variável que faz a diferenciação do quanto importante esse conhecimento pode ser. A lista com os termos(informação sobre o assunto) desse conhecimento é armazenado no atributo *ArrayList<Valor> listaValoresParaEsseNodoOntologia*. O termo é representado pela classe *Valor* e esse possui o atributo *DescricaoValor* com a descrição. *QuantidadeRepeticao* é o qualificador do quanto o agente conhece e escreveu sobre o termo e o atributo *boolean ConsultaRecomendacaoOk* é utilizado como controle das recomendações já realizadas.

Os interesses que um usuário possui ao participar do portal está sendo armaze-

nado na classe *InteressesUsuario*. Essa possui uma lista de *Interesse* que é definido no método *ArrayList<Interesse> ListaInteresses*. Essa lista é utilizada para o cálculo de recomendação de novos usuários para uma comunidade e utilizada para resposta de mensagem do agente usuário para a comunidade.

Os interesses de uma comunidade estão representados na classe *InteressesComunidade* no atributo do tipo lista *ArrayList<Interesse> ListaInteresses*, essa informação é utilizada para determinar a intersecção entre as comunidades pela similaridade dessa informação entre elas.

O tipo *Interesse* é a classe com o mapeamento do interesse de um usuário ou comunidade. O atributo *IdOntologia* define o tipo de interesse (identificador na ontologia para assim as informações terem equivalência e ser a respeito do mesmo assunto), sendo que *ValorPesoOntologia* é o atributo que faz a diferenciação do quanto importante esse interesse pode ser e os termos de interesse estão representados pela lista *ArrayList<ValorInteresse> ValoresInteresse*.

E o *ValorInteresse* é o objeto que armazena os termos de interesse do agente. Sendo o atributo *Valor* a descrição do termo e o quanto o agente tem interesse nesse assunto é definido pelo atributo *Qualificador*. Esse também é utilizada no cálculo de recomendação.

4.1.5 Classes de percepção

As classes de percepção na implementação correspondem aos comportamentos do Agente para receber ou enviar mensagens. Para um agente disponibilizar essa forma de interação com o ambiente é necessário que extenda a uma classe da plataforma Jade do tipo *Behaviour*. As classes do Sistema extendem ao tipo *CyclicBehaviour*. Esse tipo é uma thread que fica disponível para receber mensagens durante todo o tempo de vida do agente. Com isso os comportamentos estarão disponíveis para receber mensagens de outros agentes em todo o seu ciclo de vida. As classes de percepção são:

- *PercepcaoInterseccaoEntreAsComunidade* possui os métodos com a funcionalidade exposta para o ambiente, faz a comunicação com outros agentes comunidade através de troca de mensagens para localizar comunidades com interesses em comum. Essa classe contém o método *InterseccoesEntreComunidade* contém a rotina de envio de mensagens aos agentes comunidade com o interesse para a consulta e cálculo de intersecção. O método *action* fica esperando o agente receber uma mensagem, isso na template(filtro) apropriada para essa percepção. Para mensagens de *request* é verificado na base de conhecimento o interesse similar e enviado como retorno ao solicitante, e nas mensagens de *response* será executado a capacidade de intersecção entre as comunidades origem e a que enviou a mensagem de retorno.

- *PercepcaoManutencaoPerfilComunidade* é a percepção da comunidade que atualiza o perfil e o comportamento é executado pelo método *action* que recebe uma mensagem com a template de atualização e executa a capacidade de atualização do perfil.

- *PercepcaoDeManutencaoPerfilUsuario* possui o método com a funcionalidade de atualização do perfil. A *action* recebe a mensagem com a solicitação e executa a capacidade de atualização do perfil.

- *PercepcaoUsuariosPotencias_Novos* possui os métodos com as regras da funcionalidade que localiza novos usuários para a comunidade. O método *UsuariosPontencias_Novos* é a funcionalidade que envia aos agentes usuário que participam do grupo de comunidade a solicitação com o interesse da comunidade com o objetivo de recomendar novos usuários. A mensagem que o usuário recebe é atráves do método *action*, para as mensagens com *template* de *request* é consultado na base o interesse similar e retorna ao agente comunidade uma mensagem de *response*, o agente comunidade recebe a mensagem de *response* e executa a capacidade que determina o grau de interesse da comunidade no usuário.

- *PercepcaoVerificacaoInterseccao* é o comportamento que envia para os agentes usuário da comunidade uma mensagem com o conhecimento que precisa de recomendação de agente com conhecimento similar. O método *InterserccaoEncontrada* cria e envia a mensagem de solicitação para os agentes que participam de alguma comunidade do agente origem. A consulta de agentes que participam da comunidade é realizada pelo código

```
...
    DFAgentDescription template = new DFAgentDescription();
    ServiceDescription sd = new ServiceDescription();
    sd.setType(TipoAgente[j]);//o TipoAgente é a comunidade que ele participa
    template.addServices(sd);
    DFAgentDescription[] result = DFService.search(myAgent, template);
...
```

e o envio das mensagens é realizada pelo código

```
Object retornoOntology = Sistema.Instancia.getMinhaOntologia().ConsultaOntologia(pConhecimento);
Conhecimento conhecimentoRetorno = (Conhecimento) retornoOntology;
...
ACLMessage sendMes = new ACLMessage(ACLMessage.REQUEST);
sendMes.setOntology(InterseccaoOntology);
...
Message1 request = new Message1();
ConhecimentoAgente con = new ConhecimentoAgente();
con.adicionarConhecimento(conhecimentoRetorno);
request.setRequest(con);
sendMes.setContentObject(request);
```

```
...
myAgent.send(sendMes);
...
```

O agente recebe essa mensagem pelo método *action*, as mensagens de *request* são respondidas com os dados de conhecimento similar ao da solicitação e para as mensagens de *response* é executado a capacidade de intersecção entre o usuário com a informação do agente que retornou a mensagem.

```
public void action()
{
    MessageTemplate mInterseccao = MessageTemplate.MatchOntology(InterseccaoOntology);
    ACLMessage msg1 = myAgent.blockingReceive(mInterseccao);
    if (msg1 != null)
    {
        if (msg1.getPerformative() == ACLMessage.REQUEST)
        {
            ... O Agente recebe a solicitação de interesecção de outro agente.
                Busca conhecimento similar e retorna a mensagem
        }
        else if (msg1.getPerformative() == ACLMessage.PROPOSE)
        {
            ... Para o retorno executa a capacidade de calcular a interseção
                entre os agentes
        }
    }
}
```

4.1.6 Classes com o plano de execução

O plano de execução é a classe com as regras, sequência e validações de uma funcionalidade. Uma funcionalidade poderá ter mais de um plano; já o inverso não é correto afirmar. As classes do Sistema que se caracterizam como plano são:

- *InterseccaoComunidade* é o plano com as regras de intersecção entre comunidades. O método *IniciaInterseccoesEntreComunidade* localiza a informação que precisa ser enviada para os outros agentes e chama o serviço que cria a mensagem com o interesse e os envia para os outros agentes. Para calcular a intersecção entre os interesses de duas comunidades é utilizado o método *InterseccoesEntreComunidade*, ele utiliza o Algoritmo 10 que está implementado na classe *CalculoAlgoritmo* para determinar o grau de interesse entre o agente origem e retorno.

- *InterseccaoUsuario* é o plano com as regras de intersecção entre usuário. O método *IniciaInterserccaoEncontrada* localiza a informação que precisa ser enviada para os outros agentes e chama o serviço que cria a mensagem com o interesse e os envia para os outros agentes. Para calcular a intersecção entre os interesses de dois usuários é utilizado o método *InterseccoesEntreComunidade*,

ele utiliza o Algoritmo 10 que está implementado na classe *CalculoAlgoritmo* para determinar o grau de interesse entre o agente origem e retorno. Quando um agente recebe um pedido de intersecção verifica com a utilização do método *GetConhecimentoParecido*, nesse é analisado na base de crença verificando se existe informação para o termo solicitado.

- *ManutencaoComunidade* é o plano que executa as validações para atualização de um dado. Se um dado não existe na base de crença o mesmo é adicionado e aos que existe é atualizado somente o atributo que qualifica o quanto é o interesse da comunidade no termo, essa validação é feita pelo método *AtualizarPerfilComunidade*.

- *ManutencaoUsuario* é o plano que executa as validações para atualização de conhecimento ou interesse do usuário. Quando um termo não existe na base de crença o mesmo é adicionado e aos que existe é atualizado somente o atributo que qualifica o quanto é o conhecimento ou interesse do usuário nesse termo, essa validação é feita pelo método *AtualizaPerfilUsuario*.

- *UsuariosPotenciaisNovos* é o plano que procura analisar e solicitar a recomendação de usuários para a comunidade. Esse plano contém o método *IniciaUsuariosPontencias_Novos* que é executado pelo agente comunidade, aqui é verificado o interesse de busca de novos usuários, criado e enviado a mensagem para todos os agentes que não participam da comunidade e fazem parte do grupo de intersecção da comunidade. O agente que recebe esse mensagem é o agente usuário e o mesmo analisa essa solicitação através do método *GetInteresseParecido* e envia o resultado para o solicitante. Esse retorno é recebido pelo agente comunidade e executa do método *UsuariosPontencias_Novos* que faz o cálculo de recomendação do usuário para a comunidade de acordo com o grau de similaridade.

4.1.7 Algoritmos para Recomendação de Usuários e Comunidades

A classe *CalculoAlgoritmo* possui os algoritmos de cálculo de recomendação de usuários, comunidades e novos usuários para a comunidade. O método *CalculoInteresseEmOutroUsuario* é baseado no Algoritmo 1 da especificação do sistema e procura um valor correspondente a similaridade de conhecimento entre dois agentes usuários. Abaixo o algoritmo correspondente em java.

Algorithm 9: CalculoInteresseEmOutroUsuario

```
1  public double CalculoInteresseEmOutroUsuario(ConhecimentoAgente ConhecimentoOrigem,
2                                                ConhecimentoAgente ConhecimentoRetorno)
3  {
4      double Score = 0;
5      for (Conhecimento origem : ConhecimentoOrigem.getListaConhecimentos())
6      {
7          for (Conhecimento retorno : ConhecimentoRetorno.getListaConhecimentos())
8          {
9              if (origem.getIdOntologia().equalsIgnoreCase(retorno.getIdOntologia()))
10             {
11                 double ScoreAux = 0;
12                 for (Valor valor : origem.getListaValoresParaEsseNodoOntologia())
13                 {
14                     for (Valor valorOutro : retorno.getListaValoresParaEsseNodoOntologia())
15                     {
16                         if (valor.getDescricaoValor().equalsIgnoreCase(
17                             valorOutro.getDescricaoValor()))
18                         {
19                             ScoreAux += valorOutro.getQuantidadeRepeticao();
20                             break;
21                         }
22                     }
23                 }
24                 if (retorno.getPesoNaOntologia() > 0)
25                     Score += (ScoreAux * origem.getPesoNaOntologia());
26                 else
27                     Score += ScoreAux;
28                 break;
29             }
30         }
31     }
32     return Score;
33 }
```

O *CalculoInteresseEmOutraComunidade* é método é baseado no Algoritmo 2 da especificação do sistema e procura um valor correspondente a similaridade de interesse entre dois agentes comunidade, esse algoritmo em java é representado na sequência.

Algorithm 10: CalculoInteresseEmOutraComunidade

```
1  public double CalculoInteresseEmOutraComunidade(InteressesComunidade InteressesOrigem,
2                                                   InteressesComunidade InteresseRetorno)
3  {
4      double Score = 0;
5      for (Interesse origem : InteressesOrigem.getListaInteresses())
6      {
7          for (Interesse retorno : InteresseRetorno.getListaInteresses())
8          {
9              if (origem.getIdOntologia().equalsIgnoreCase(retorno.getIdOntologia()))
10             {
11                 double ScoreAux = 0;
12                 for (ValorInteresse valor : origem.getValoresInteresse())
13                 {
14                     for (ValorInteresse valorOutro : retorno.getValoresInteresse())
15                     {
16                         if (valor.getValor().equalsIgnoreCase(valorOutro.getValor()))
17                         {
18                             ScoreAux += valorOutro.getQualificador();
19                             break;
20                         }
21                     }
22                 }
23                 if (retorno.getValorPesoOntologia() > 0 && ScoreAux > 0)
24                     Score += (ScoreAux * origem.getValorPesoOntologia());
25                 else
26                     Score += ScoreAux;
27                 break;
28             }
29         }
30     }
31     return Score;
32 }
```

O método *CalculoNovosUsuariosComunidade* é baseado no Algoritmo 3 da especificação do sistema e procura um valor correspondente ao quanto um usuário tem interesses e pontencial para integrar a comunidade. Esse algoritmo em java fica assim

Algorithm 11: CalculoNovosUsuariosComunidade

```
1  public double CalculoNovosUsuariosComunidade(InteressesComunidade InteressesOrigem,
2                                                InteressesUsuario InteresseRetorno)
3  {
4      double Score = 0;
5      for (Interesse origem : InteressesOrigem.getListaInteresses())
6      {
7          for (Interesse retorno : InteresseRetorno.getListaInteresses())
8          {
9              if (origem.getIdOntologia().equalsIgnoreCase(retorno.getIdOntologia()))
10             {
11                 double ScoreAux = 0;
12                 for (ValorInteresse valor : origem.getValoresInteresse())
13                 {
14                     for (ValorInteresse valorOutro : retorno.getValoresInteresse())
15                     {
16                         if (valor.getValor().equalsIgnoreCase(valorOutro.getValor()))
17                         {
18                             ScoreAux += valorOutro.getQualificador();
19                             break;
20                         }
21                     }
22                 }
23                 if (retorno.getValorPesoOntologia() > 0 && ScoreAux > 0)
24                     Score += (ScoreAux * origem.getValorPesoOntologia());
25                 else
26                     Score += ScoreAux;
27                 break;
28             }
29         }
30     }
31     return Score;
32 }
```

4.2 Diagramas de Sequência

Os diagramas utilizados tem como objetivo auxiliar no entendimento sobre a sequência de execução das regras mais complexas e ver como funciona a relação entre as classes apresentadas. Não será detalhado todos os diagramas, com base nos diagramas é possível entender a relação das outras funcionalidades, já que a estrutura é a mesma, só muda o nome dos métodos envolvidos nas chamadas dessas relações.

É necessário explicar a relação do agente comunidade, quando recebe a solicitação de intersecção e que métodos utiliza para chegar ao retorno e por quais classes passa para obter esse resultado. A Figura 4.3 mostra como o ambiente/portal se comunica com o agente comunidade$_O$. O sistema possui uma interface *IFuncoesEx-*

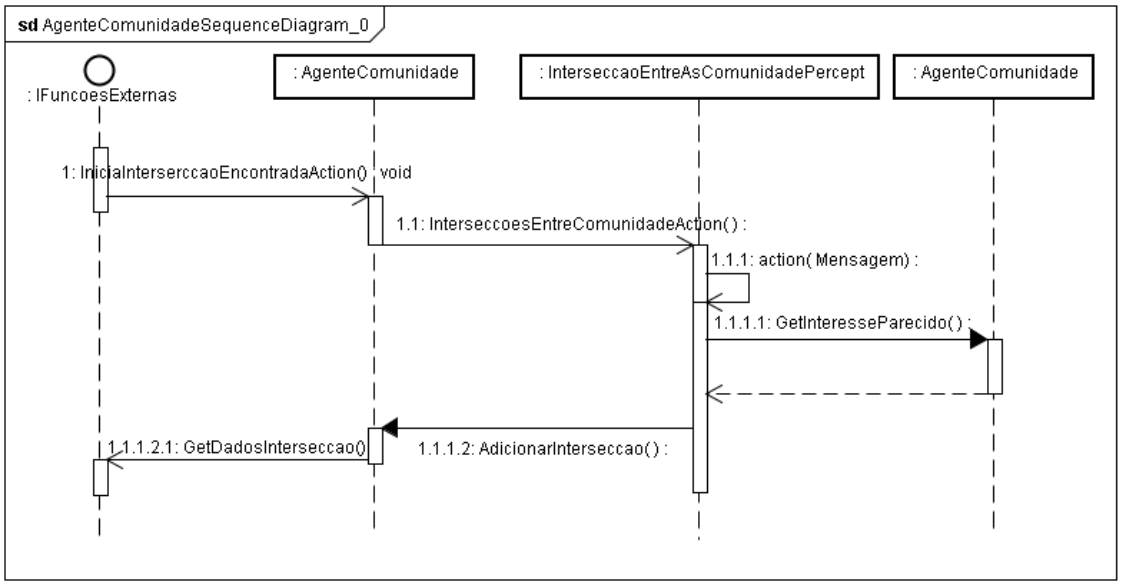

Figura 4.3: Diagrama de Sequência - Intersecção com o Ambiente

ternas com as funcionalidades para um sistema de outro ambiente se comunicar. A classe que implenta essa interface servirá como *middleware* com o portal e nela será recebido a solicitação, mapeando os dados para estrutura correta do Sistema e após, chama $AgenteComunidade_O$ com o método a ser executado. Ele só conseguirá se comunicar com outros agentes através da troca de mensagem e essa funcionalidade se encontra na classe $PercepcaoInterseccaoEntreAsComunidade_O$. Na percepção é criada a mensagem que é enviada para os agentes e após o envio, ela é recebida pela mesma classe, porém na instância do $agente_D$. Com os dados de *request* é solicitado ao agentes os interesses parecidos, que são enviados para o $agente_O$. Com esse retorno O calcula a intersecção, adiciona na sua lista de intersecções e a mesma é retornada ao ambiente pela interface.

A execução da capacidade de verificação de intersecção entre comunidades está descrito no diagrama da Figura 4.4, com a relação entre as classes com a sequência de execução, levando em consideração o agente, capacidade, plano e a classe com o cálculo de recomendação.

A execução de uma solicitação de recomendação de novos usuários para a comunidade esta demosntrada na Figura 4.5. O ambiente/portal se comunica com o $agente\ comunidade_O$ através da interface *IFuncoesExternas*, que serve como *middleware* com o portal, recebe a solicitação, mapeando os dados para estrutura correta do Sistema e chama $AgenteComunidade_O$ com o método a ser executado. Na execução é enviado uma mensagem com essa solicitação para os agentes usuários $_D$ através da percepção $PercepcaoUsuariosPotencias\_Novos_O$. A mensagem é recebida pelo agente $_D$ e faz a verificação de interesses similares, retornando como *response* a mensagem do solicitante. Com essa mensagem o $agente_O$ calcula a intersecção e

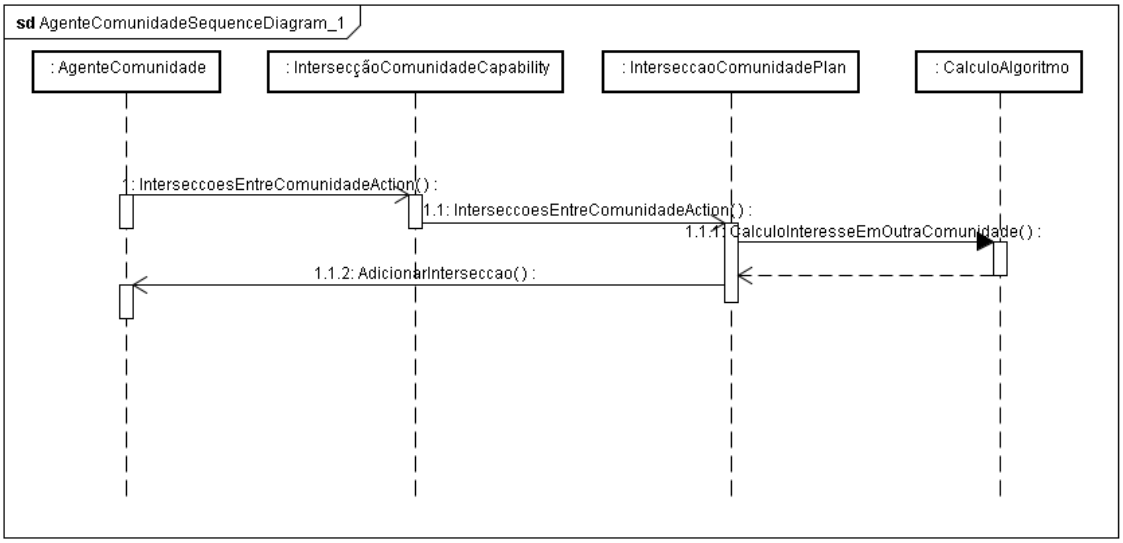

Figura 4.4: Diagrama de Sequência - Relação entre as classes

adiciona na sua lista de novos usuários e retorna ao ambiente pela interface com o resultado da recomendação.

A execução da capacidade de consultar novos usuários para a comunidade é descrito no diagrama da Figura 4.6, com a relação entre as classes com a sequência de execução, levando em consideração o agente, capacidade, plano e a classe com o cálculo de recomendação.

A Figura 4.7 mostra como o ambiente/portal se comunica com o agente usuário $_O$. O sistema possui uma interface *IFuncoesExternas* com as funcionalidades para um sistema de outro ambiente se comunicar. A classe que implementa essa interface servirá como *middleware* com o portal e nela será recebido a solicitação, mapeando os dados para estrutura correta do Sistema e após, chama $AgenteUsuario_O$ com o método a ser executado. Ele só conseguirá se comunicar com outros agentes através da troca de mensagem e essa funcionalidade se encontra na classe $PercepcaoVerificacaoInterseccao_O$. Na percepção é criada a mensagem que é enviada para os agentes e após o envio, ela é recebida pela mesma classe, porém na instância do agente $_D$. Com os dados de *request* é solicitado ao agentes os conhecimentos parecidos, que são enviados para o agente$_O$. Com esse retorno O calcula a intersecção e adiciona $_D$ na sua lista de intersecções e a mesma é retornada ao ambiente pela interface.

A execução da capacidade de verificar interesecção entre usuários no diagrama da Figura 4.8, com a relação entre as classes com a sequência de execução, levando em consideração o agente, capacidade, plano e a classe com o cálculo de recomendação.

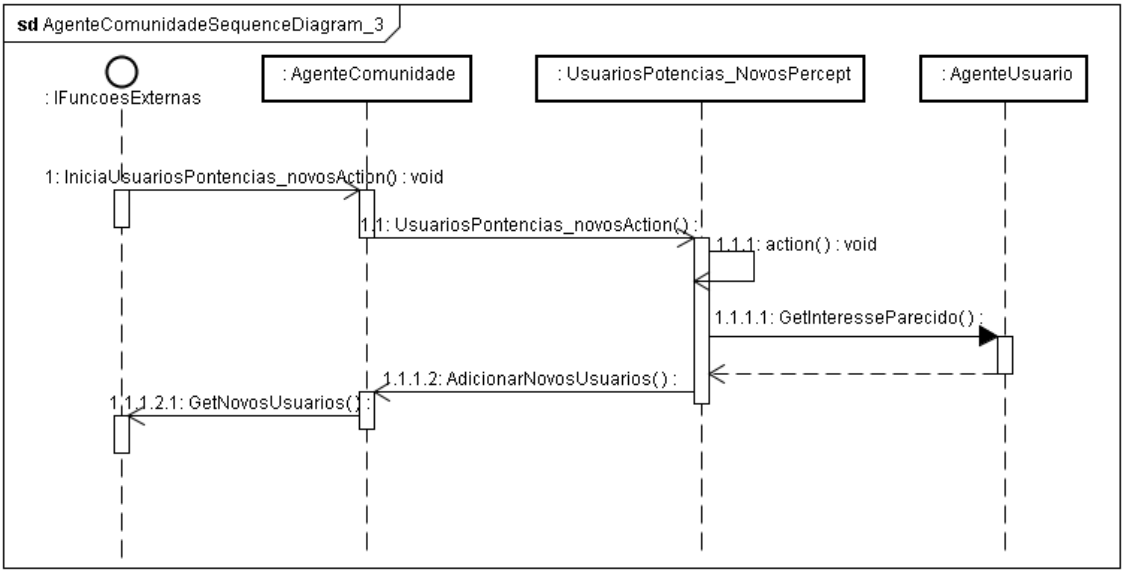

Figura 4.5: Diagrama de Sequência

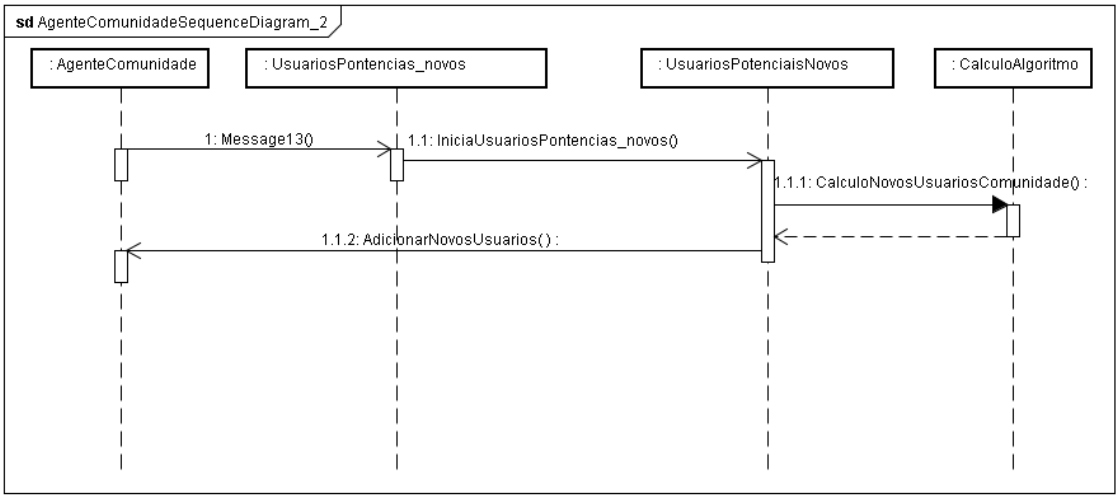

Figura 4.6: Diagrama de Sequência

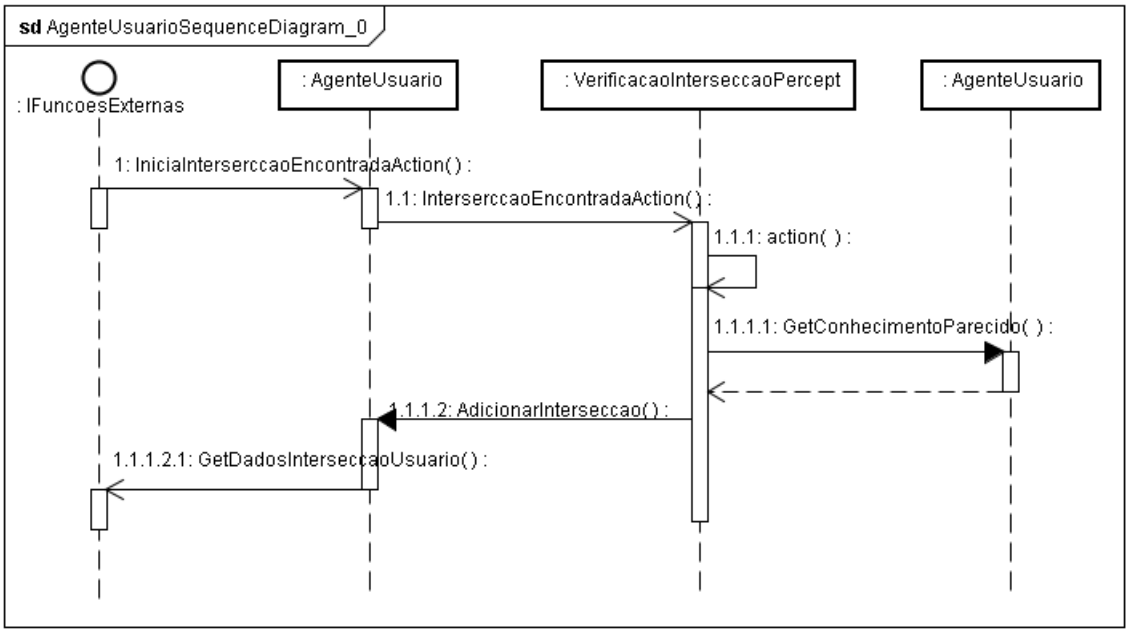

Figura 4.7: Diagrama de Sequência

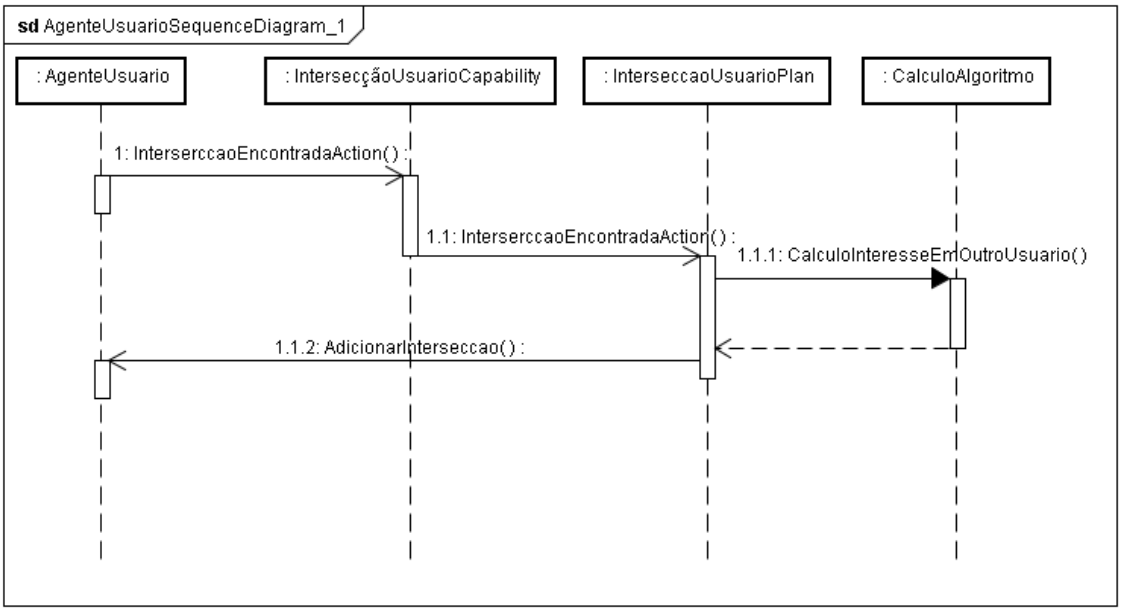

Figura 4.8: Diagrama de Sequência

4.3 Considerações Finais

As classes descritas nesse capítulo são as principais no Sistema. Foi apresentado as classes que correspondem aos agentes(*AgenteUsuario*, *AgenteComunidade*) e seus métodos, as classes de capacidades, planos e a estrutura de dados que foi utilizada na implementação. E para exemplificação da relação entre as mesmas foi utilizado diagramas de classes e de sequência.

No capítulo seguinte, será demonstrado com o auxílio de alguns cenários, como é o comportamento dos agentes e resultados optidos na simulação das principais funcionalidades.

5 CENÁRIOS DE USO

No capítulo anterior foi descrito a implementação do Sistema. Nesse capítulo será feita a validação do Sistema proposto com cenários projetados e com as principais relações entres os agentes para se validar a teoria de como é a estrutura de perfil do usuário e comunidade visando simular os comportamentos dos agentes. Para isso será demonstrado os cenários de intersecção entre usuários, comunidades e recomendação de novos usuários para uma comunidade. O perfil das comunidades e usuários será representado em formato XML, de acordo com a estrutura de dados utilizada no sistema e os mesmos estão incluídos neste trabalho como anexo.

As crenças do agente, representando os interesses do usuário e os interesses da comunidade são privativas. Para determinar as recomendações, os usuários e comunidades não tem acesso a todo o perfil de quem responde uma solicitação. A única informação disponível é o retorno com apenas as informações similares a requisição. Com isso um agente não expõem em nenhum momento todo o seu perfil de interesse.

Os agentes utilizados nos cenários serão os agentes $Ag01, Ag02, Ag03, Ag04, Ag05$ e as comunidades $Com01, Com02$ e $Com03$. Na Tabela 5.1 são ilustrados os agentes utilizados na explicação dos cenários. A primeira e segunda coluna tem o nome do agente e comunidades que o mesmo participa. A terceira contém a lista de agentes do tipo usuário que estão aptos a receber a mensagem de intersecção e a quarta possui a lista de comunidades das quais o usuário pode receber uma mensagem solicitando a similaridade de interesse, com isso a comunidade pode avaliar o grau de interesse nesse usuário.

Tabela 5.1: Relação de Usuário e Comunidade

Usuário	Comunidade	Msg para Usuário	Msg da Comunidade
$Ag01$	$Com01$	$Ag02, Ag03, Ag04$	$Com02, Com03$
$Ag02$	$Com01, Com02$	$Ag01, Ag03, Ag04$	$Com03$
$Ag03$	$Com01, Com02$	$Ag01, Ag02, Ag04$	$Com03$
$Ag04$	$Com01$	$Ag01, Ag02, Ag03$	$Com02, Com03$
$Ag05$	$Com03$		$Com01, Com02$

Para os experimentos conduzidos neste Capítulo, definimos os agentes usuário e

comunidade possuindo interesses sobre o domínio de Linguagens de Programação. A Figura 5.1 representa o ”perfil estático” dos agentes usuário e comunidades. Neste perfil, os agentes manifestam ”interesses” sobre programação em algumas linguagens.

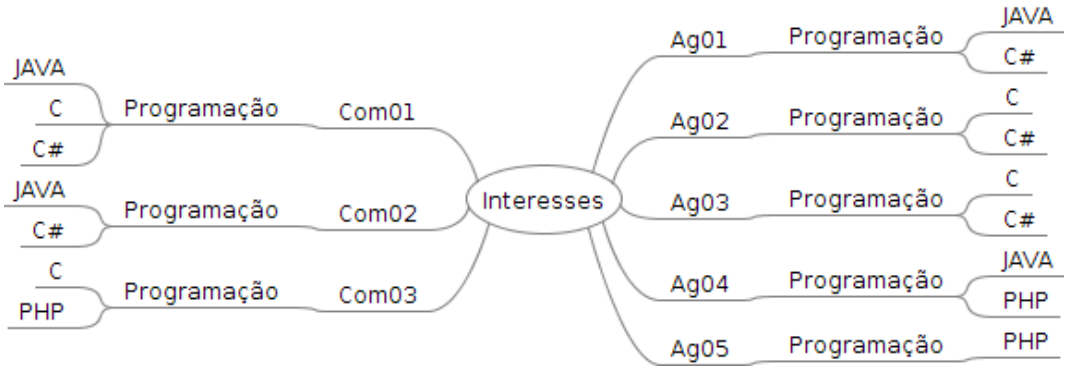

Figura 5.1: Perfil contendo os interesses dos agentes usuário e comunidade.

Para o conhecimento interno, definimos os cinco agentes contendo em sua base de crenças informações sobre o domínio de Linguagens de Programação. A Figura 5.2 ilustra as crenças de cada um dos agentes sobre o domínio. Por exemplo, o agente $Ag01$ possui conhecimentos específicos sobre as linguagens JAVA, C, C# e PHP. Em cada uma das linguagens, o agente manifestou interesse sobre alguns termos, através de alguma ferramenta de interação no Portal. Por exemplo, ele possui interesses ou dúvidas a respeito do uso de *Buttons* em JAVA e C#. Os números entre parênteses simbolizam a frequência de uso dos termos nas ferramentas de interação. O perfil de cada usuário e comunidade, com os interesses e conhecimentos é informado ao agente pelo portal, esse é o responsável por classificar e qualificar os dados.

Para verificar se o comportamento está de acordo com a especificação, serão utilizadas duas telas protótipadas, desenvolvidas em JAVA com o único objetivo de permitir a análise do comportamento. Elas não fazem parte do projeto, já que o portal não utilizará essa parte gráfica e sim o comportamento e informações disponibilizadas e enviadas para o agente.

A tela para a simulação do comportamento do usuário é a representação da Figura 5.3 e está divida em áreas. A área ***A*** corresponde aos interesses do usuário; a área ***B*** representa o conhecimento do usuário; a área ***C*** contém os novos agentes indicados pela intersecção; a área ***D*** é o log de mensagem recebida e a área ***E*** contém a lista das comunidades que o agente participa.

O comportamento da comunidade está representado na tela da Figura 5.4 e está divida em áreas. A área ***A*** corresponde ao perfil da comunidade; a área ***B*** representa a lista de novos usuários recomendados para a comunidade; a área ***C*** possui os novos agentes recomendados por intersecção; a área ***D*** apresenta o log de mensagens recebidas e a área ***E*** mostra os usuários que participam da comunidade.

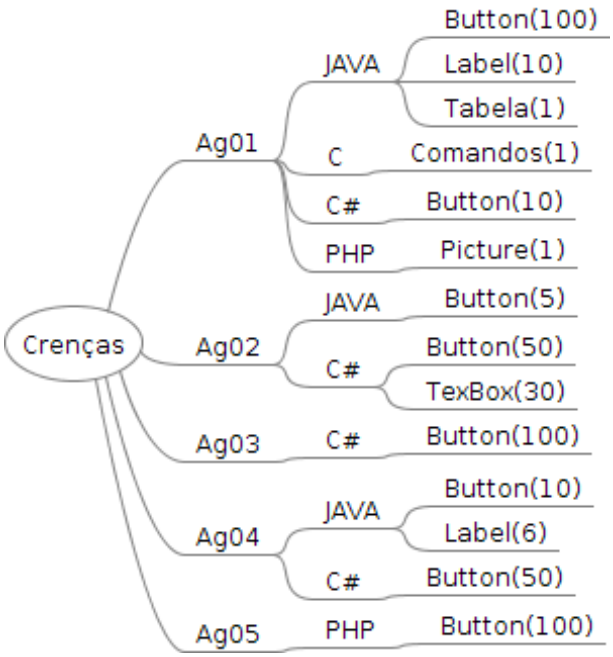

Figura 5.2: Base de Crenças dos agentes.

5.1 Cenário 1

Nesse cenário é feita a simulação de recomendação de usuários. Para isso o agente *Ag01* representa o agente origem, que solicita uma consulta de intersecção para validar sua base de crenças. Esta consulta inicia com mensagens disparadas apenas para os agentes que fazem parte de sua comunidade (*Com01*). Quando um agente participa de várias comunidades, todos os agentes das comunidades relacionadas são considerados. Os agentes aptos a receber e responder essa mensagem são os agentes (*Ag02, Ag03, Ag04*). A Figura 5.5 ilustra, no campo $\boldsymbol{C}$, os resultados calculados dos agentes apresentando as similaridades quanto ao interesse em seus perfis.

Para essa simulação podemos confirmar que, para a identificação da ontologia *Java* e o termo *Label* somente o *Ag04* respondeu. Para o termo *Button* os agentes *Ag02* e *Ag04* responderam e estão ordenados pelo valor que cada um conhece. E a indentificação da ontologia *C#*, termo *Button* os agentes que responderam foram *Ag02*, *Ag03*, *Ag04*. Não houve resposta para as ontologia(termo) *Java(Tabela)*, *C(Comandos)*, *PHP(Picture)* pois entre os usuários que participam da comunidade *Com01*, única que o *Ag01* participa, não tem outro usuário que possui conhecimento sobre essas informações.

O cálculo desse grau de interesse é feito pelo Algoritmo 9, nesse caso o conhecimento dos agentes em *C#(BUTTON)* é $Ag02(50)$, $Ag03(100)$ e $Ag04(50)$, ordenando esses valores temos que o $Ag03$ é o primeiro da lista de recomendação para o $Ag01$, fato comprovado na $\boldsymbol{C}$ da figura mencionada acima que contém o índice de recomendação. No cálculo mostrado o peso do quanto importante essas informações são na

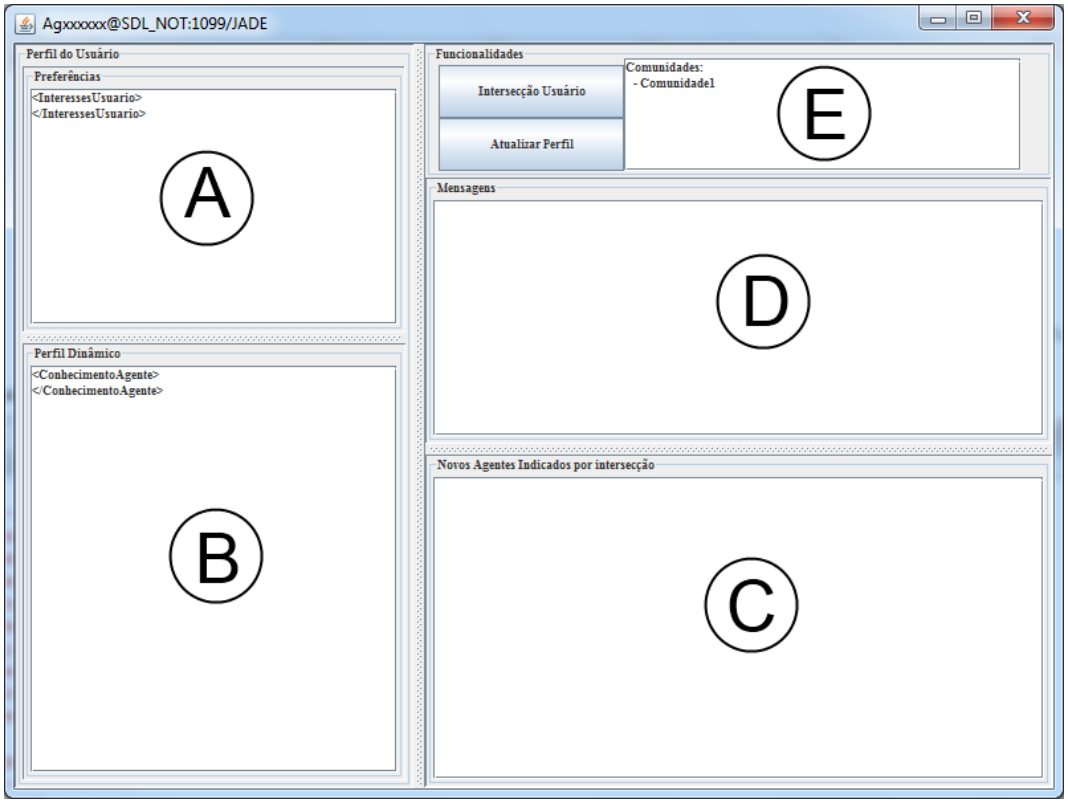

Figura 5.3: Frame Agente Usuário

Ontologia é de 1, com isso não é alterado a ordem. Mas em outros casos o peso na Ontologia serve como um fator de harmonização do conhecimento de termos de cada identificação de ontologia, por exemplo o valor 50 que o $Ag02$ conhece é multiplicado pelo valor de peso que tem na ontologia em que esse conhecimento está vinculado.

5.2 Cenário 2

Nesse cenário é demonstrada a simulação de intersecção entre comunidades, onde a comunidade *Com01* solicita para os termos de interesse, a consulta de intersecção. Todos os agentes do tipo *AgenteComunidade* interagem entre si usando os termos de solicitação de interesse. Na Figura 5.6 está ilustrado, no campo ***C***, os resultados calculados de agentes com similaridade de interesse. Nesse resultado podemos verificar que os agentes $Com02$ e $Com03$ responderam à solicitação para os termos *C#, Java, C*. Com o resultado podemos verificar que o agente $Com03$ tem relação com o agente $Com01$ porque possue o mesmo interesse em ***PROGRAMACAO(C)***. A comunidade $Com02$ faz intersecção com a comunidade $Com01$ pelos interesses em ***PROGRAMACAO(C#)*** e em ***PROGRAMACAO(JAVA)***.

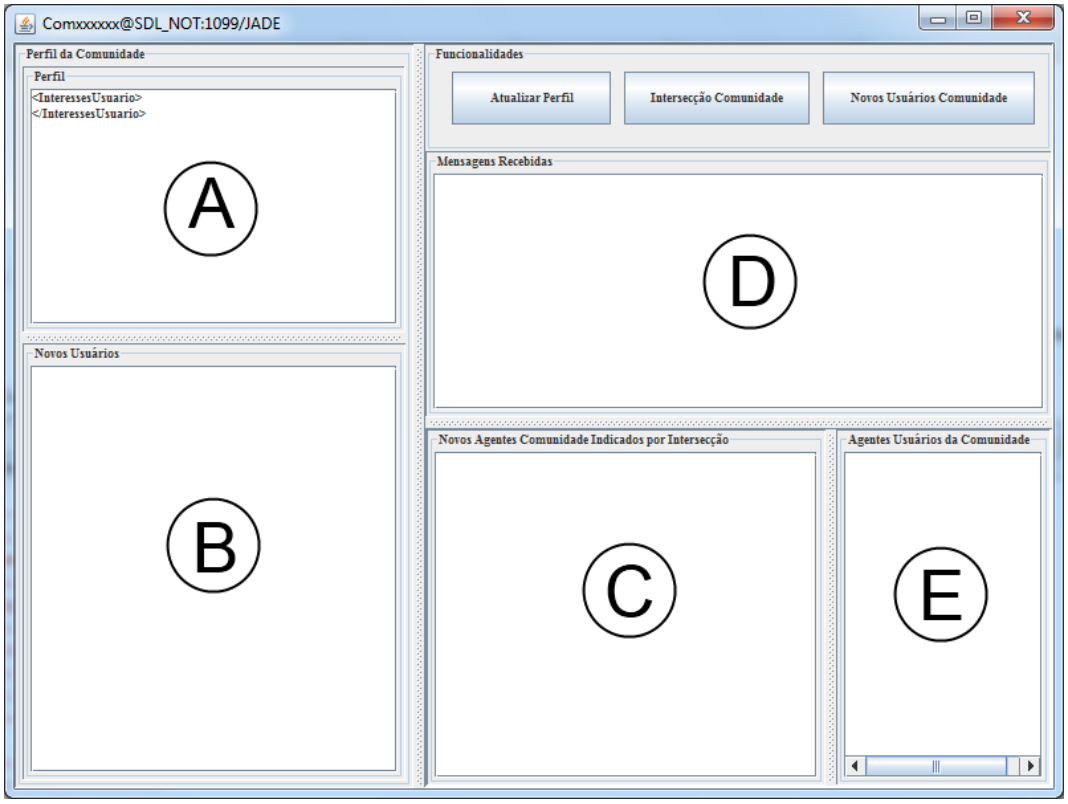

Figura 5.4: Frame Agente Comunidade

5.3 Cenário 3

Este cenário demonstra a possibilidade de recomendar novos usuários para a comunidade. Nesse exemplo a comunidade *Com03* envia uma mensagem para os usuários que participam das comunidades (*Com01* e *Com02*), contendo os interesses da comunidade. Os agentes verificam se tem alguma similaridade de interesse e só respondem quando possuem e, com esse retorno, é calculado o interesse da comunidade neste usuário. O resultado para esse cenário com a recomendação está ilustrado na Figura 5.7, no campo ***B***.

A comunidade *Com03* tem interesse em *C* e *PHP*, com isso os agentes que possuem o mesmo interesse na simulação de perfis apresentados são os agentes *Ag02* e *Ag03* na ontologia(termo) *Programacao(C)*, e *Ag01* em *Programacao(PHP)*. Nesse caso somente esses são usuários com capacidade de integrar a comunidade *Com03*.

5.4 Considerações Finais

O comportamento dos agentes nos principais cenários foi de acordo com a funcionalidade implementada. Esses cenários são os que poderiam não ter comportamentos de acordo com a especificação. Com o resultado obtido é possível constatar que da forma que o conhecimento foi mapeado, a lógica para recomendação funciona e irá

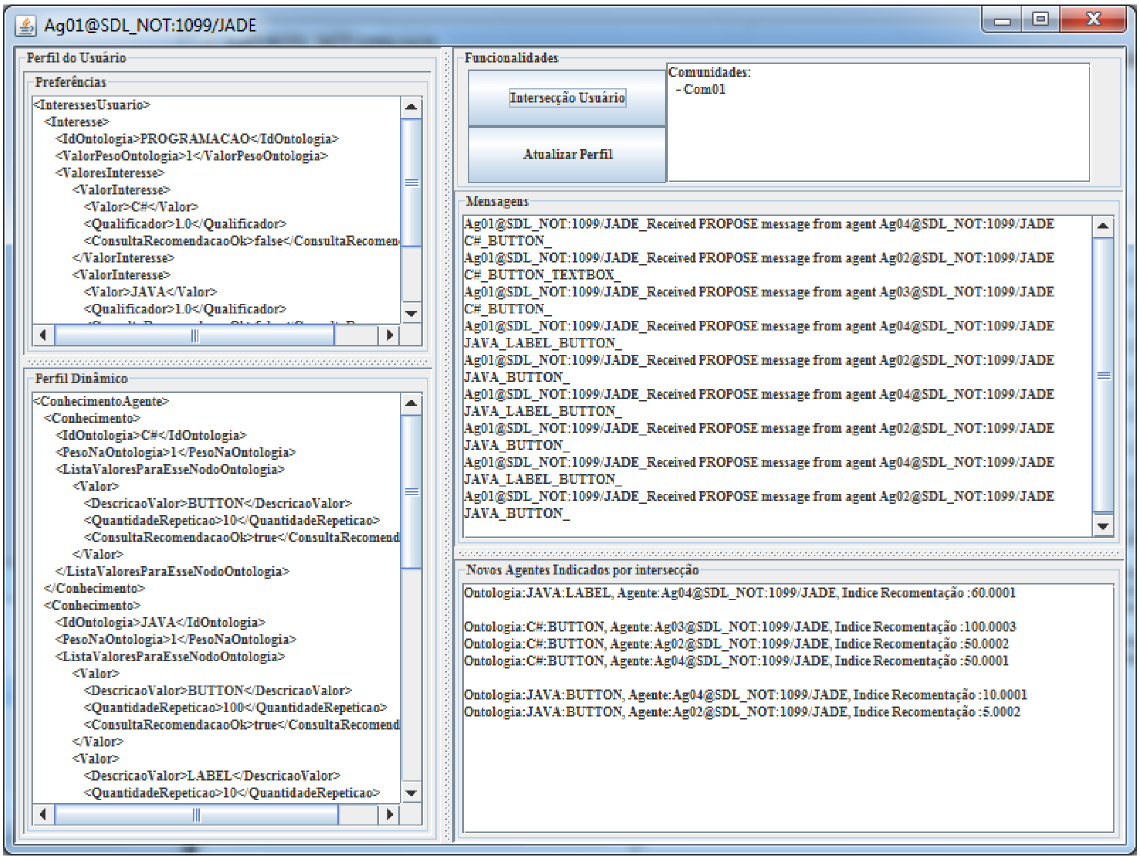

Figura 5.5: Resultado Cenário 1

proporcionar aos agentes recomendações úteis.

Quanto a utilidade da informação, principalmente a recomendação de outros usuários, a interface do portal que o usuário utiliza pode mostrar os resultados, possibilitando adicionar na base de relacionamento os que mostrarem um maior resultado de intersecção. Além de possibilitar ao usuário executar uma nova pesquisa de intersecção em qualquer momento que ele a sim o necessitar e ainda possibilitar que essa pesquisa(recomendação) seja feita apenas para um termo específico, selecionado pelo usuário, essa funcionalidade não tem a assinatura disponível ao ambiente mas está citado em *Trabalhos Futuros*.

Por fim, embora os testes tenham sido bem sucedidos, vale lembrar que o agente só vai estar monitorando e estar disponível no ambiente no momento em que o usuário acessar o portal, com isso além das recomendações que são solicitadas pelo usuário é fundamental as rotinas de intersecção por ciclo, para que a cada x horas a rotina de recomendação seja executada e que tenha regras de execução imediata quando o mesmo ficou offline por mais tempo que x horas.

Concluíndo, acredito que a interação entre usuários, colaboração e disponibilidade de informação seja um diferencial para comunidades que utilizarem essa ideia de recomendação, tendo esse diferencial, eu como usuário do portal, participando

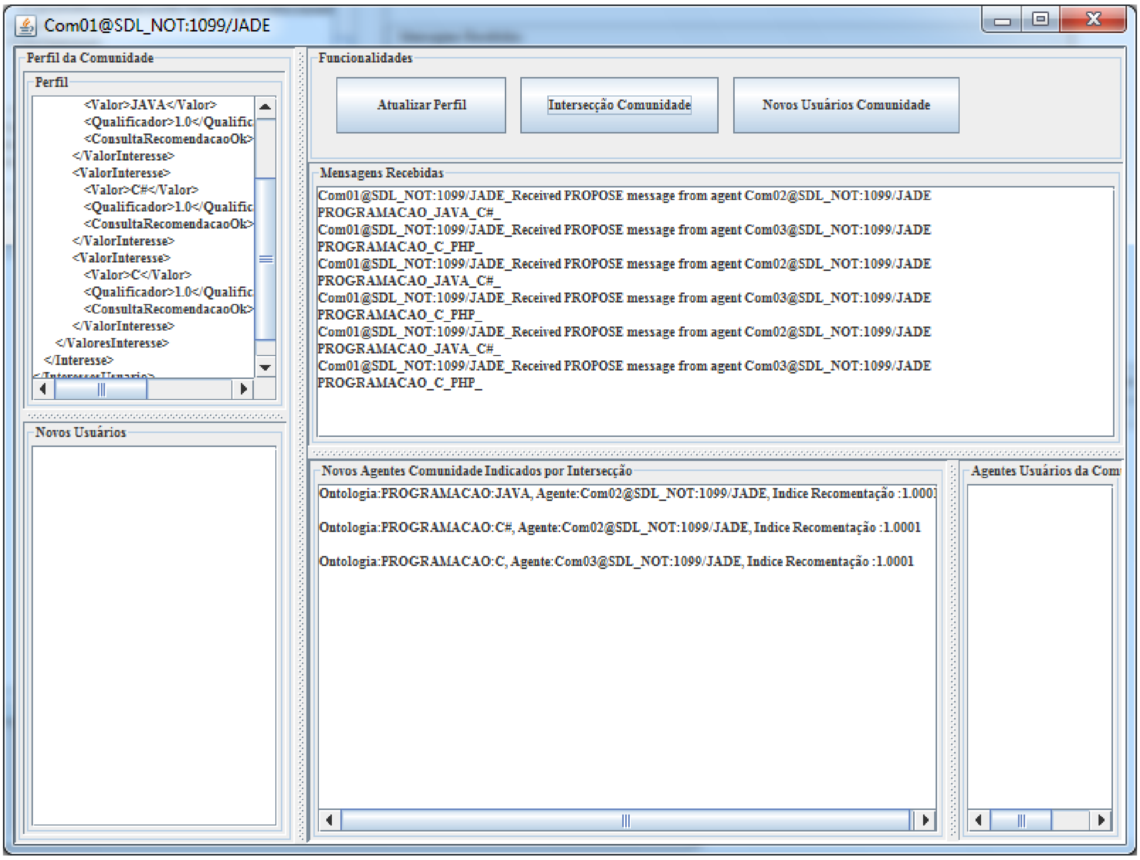

Figura 5.6: Resultado Cenário 2

de uma comunidade, no momento que precisar ou estiver escrevendo sobre algum assunto, não preciso procurar para encontrar referencial de informação já publicada no ambiente. Essa informação de outros que já escreveram a respeito, vai estar disponível, com uma prioridade de recomendação já classificada e analisada do quanto importante pode ser selecionar informação A e não a informação B.

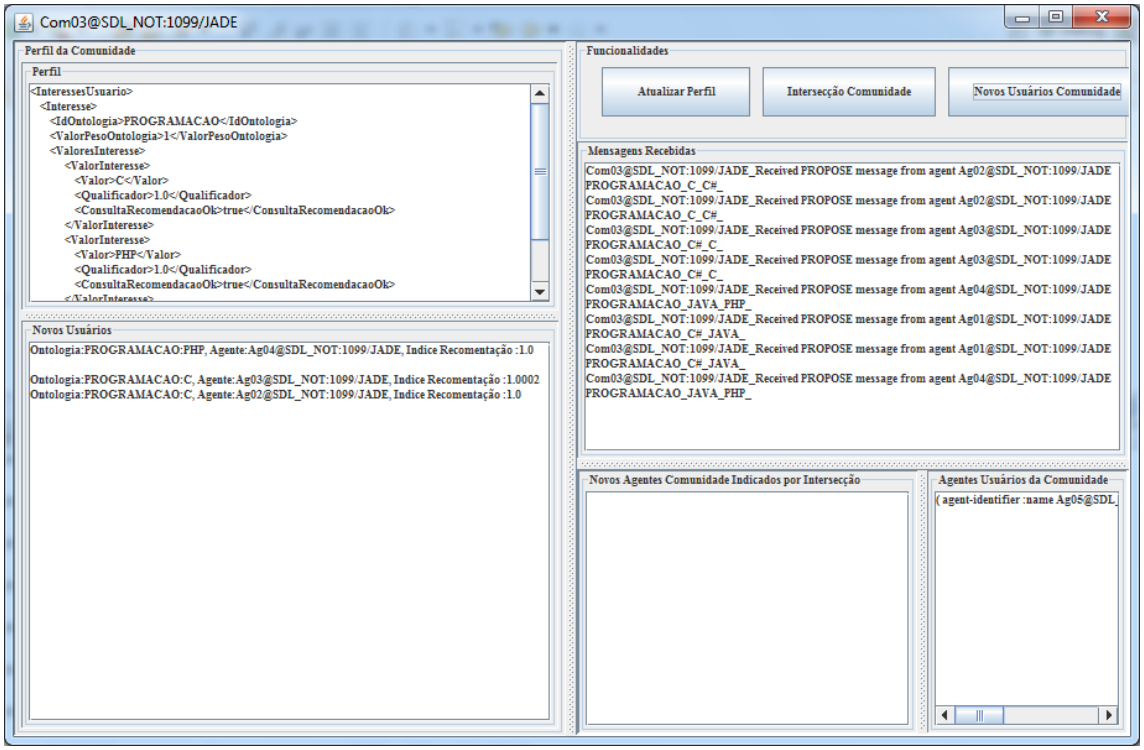

Figura 5.7: Resultado Cenário 3

6 CONCLUSÃO

No presente trabalho apresentamos a especificação, modelagem e desenvolvimento de um sistema multiagentes para compartilhamento de conhecimento em Comunidades de Prática. Focalizamos o trabalho no gerenciamento de perfis de usuário de um portal que possui comunidades e usuários, a recomendação de usuários com base no grau de similaridade de conhecimento em uma organização de dados baseada através do uso de ontologias. Para a modelagem e especificação foi utilizado a metodologia Prometheus e a implementação dos agentes com o auxílio da plataforma JADE atendeu a todas necessidades para a solução do problema.

As contribuições desse trabalho para a recomendação de agentes de informação estão vinculadas ao fato de ter sido definido e implementado agentes que podem se comunicar com outros somente compartilhando uma parte do perfil de conhecimento e interesse. Este conhecimento é utilizado para calcular intersecções e similaridades de conhecimento e determinar o grau/percentual de interesse entre os agentes para cada termo do conhecimento. Com isso é possível encontrar especialistas no assunto em cada nível de conhecimento e recomendar participantes e comunidades.

A partir deste modelo, é possível detectar recomendações com base no conhecimento que cada agente possui, mapeando-o em uma estrutura indexada por uma ontologia. Deste modo, pode-se padronizar o perfil de interesse através das categorização dos interesses dirigida por uma ontologia do domínio. No trabalho presente, foi definido critérios definidos pela quantificação de termos similares entre os agente, onde cada termo é maximizado por um peso no nodo de ontologia a qual o termo pertence.

6.1 Trabalhos Futuros

Para este trabalho, foram feitas restrições de escopo para permitir um estudo pontual na questão de recomendação de usuários e comunidades. Além das limitações, outros aspectos importantes foram percebidos durante o estudo, e por limitação de tempo e objetivo não foram desenvolvidas. Essa seção apresenta alguns temas

que podem ser desenvolvidos a partir do estudo realizado e servir como complemento do trabalho.

Podemos vislumbrar funcionalidades complementares para os agentes, utilizando a estrutura já existente, para efetivar uma recomendação, por exemplo, colocando o usuário em uma lista de "amigos" e bloqueando outros agentes que já foram constados não terem uma informação útil. Esta funcionalidade pode também bloquear ou desabilitar na lista, os agentes que não tem interesse sobre algum perfil ou conhecimento de um usuário.

E ainda funcionalidades com diferentes assinaturas para os serviços já modelados, para por exemplo, na intersecção permitir a busca de novos usuários a partir de um termo específico decidido pelo próprio usuário, opção para que ele também possa forçar a busca de novos usuários de todos os seus termos de conhecimento.

Outras melhorias possíveis focalizam a performance do acesso ao perfil do usuário. Pode-se avaliar armazenar o perfil em memória é mais viável, rápido e seguro, ou se o melhor é manter o mesmo em arquivo e os dados em memória sejam apenas bufferizados com os últimos acessos, ou acessos mais frequentes. Ainda nesse sentido, o que poderia ser implementado é arquivo em formato de tabela e utilizando índices para a pesquisa, assim poderá ser reduzido o tamanho do arquivo e ter um acesso direto aos termos de um tipo de conhecimento.

REFERÊNCIAS

BELLIFEMINE, F.; CAIRE, G.; POGGI, A.; RIMASSA, G. Jade: a software framework for developing multi-agent applications. lessons learned. **Inf. Softw. Technol.**, Newton, MA, USA, v.50, p.10–21, January 2008.

BELLIFEMINE, F.; POGGI, A.; RIMASSA, G. Jade, a fipa compliant agent framework. , 2001.

BELLIFEMINE, F.; POGGI, A.; RIMASSA, G. Jade: a fipa2000 compliant agent development environment. In: AUTONOMOUS AGENTS, 2001, New York, NY, USA. **Proceedings...** ACM, 2001. p.216–217. (AGENTS '01).

BERNY, V. M.; ADAMATTI, D. F.; COSTA, A. C. d. Desenvolvimento de agentes: uma análise da utilizacao da metodologia prometheus. In: III WORKSHOP-ESCOLA DE SISTEMAS DE AGENTES, SEUS AMBIENTES E APLICAçõES - III WESAAC, 2009. **Anais...** [S.l.: s.n.], 2009. Rio Grande, RS, Brasil. ISBN 978-85-7061-531-2.

BRESCIANI, P.; PERINI, A.; GIORGINI, P.; GIUNCHIGLIA, F.; MYLOPOULOS, J. Tropos: an agent-oriented software development methodology. **Autonomous Agents and Multi-Agent Systems**, Hingham, MA, USA, v.8, p.203–236, May 2004.

DAVIES, J.; DUKE, A.; SURE, Y. Ontoshare - an ontology-based knowledge sharing system for virtual communities of practice. , 2004.

DUBé, L.; BOURHIS, A.; JACOB, R. **The impact of structuring characteristics on the launching of virtual communities of practice**. [S.l.]: Journal of Organizational Change Management, 2005.

GIUNCHIGLIA, F.; MYLOPOULOS, J.; PERINI, A. The tropos software development methodology: processes, models and diagrams. In: AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS: PART 1, 2002, New York, NY, USA. **Proceedings...** ACM, 2002. p.35–36. (AAMAS '02).

JADE. **Java Agent Development Framework**. Disponível em: <http://jade.tilab.com/>. Acesso em: dezembro 2010.

KOCH, M.; LACHER, M. Integrating Community Services: A Common Infrastructure Proposal. , 2005.

LUGO, G. A. G. Um Modelo de Sistemas Multiagentes para Partilha de Conhecimento Utilizando Redes Sociais Comunitárias. , 2004.

NONAKA. Criação de Conhecimento na Empresa. , 2001.

PADGHAM, L.; WINIKOFF, M. **Developing Intelligent Agent Systems**. [S.l.]: Melbourne: John Wiley & Sons, 2004.

RUSSELL S. J.; IORVIG, P. Artificial Intelligence. , 1995.

SILVA, V. T. d.; MARIA, B. d.; LUCENA, C. J. P. d. A mde-based approach for developing multi-agent systems. In: THIRD WORKSHOP ON SOFTWARE EVOLUTION THROUGH TRANSFORMATIONS: EMBRACING THE CHANGE (SETRA 2006), 2006. **Proceedings...** [S.l.: s.n.], 2006. p.14. ISSN 1863-2122.

WENGER, E. **Communities of Practice - Learning, Meaning, and Identity**. Disponível em: <http://www.co-i-l.com/coil/knowledge-garden/cop/lmi.shtml>. Acesso em: dezembro 2010.

WENGER, E.; MCDERMOTT, R.; SNYDER, W. M. **Seven Principles for Cultivating Communities of Practice**. Disponível em: <hbsworkingknowledge.hbs.edu>. Acesso em: jun 2010.

WOOLDRIDGE, M. **An Introduction to Multiagent Systems**. [S.l.]: John Wiley & Sons, 2009.

ZAMBONELLI, F.; JENNINGS, N. R.; WOOLDRIDGE, M. Developing multiagent systems: the gaia methodology. **ACM Trans. Softw. Eng. Methodol.**, New York, NY, USA, v.12, p.317–370, July 2003.

7 ANEXOS

Algorithm 12: Xml de configuração do agente *Com01*

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <dataEstatico.InteressesComunidade>
4    <ListaInteresses>
5      <dataEstatico.Interesse>
6        <IdOntologia>PROGRAMACAO</IdOntologia>
7        <ValorPesoOntologia>1</ValorPesoOntologia>
8        <ValoresInteresse>
9          <dataEstatico.ValorInteresse>
10           <Valor>JAVA</Valor>
11           <Qualificador>1</Qualificador>
12           <ConsultaRecomendacaoOk>true</ConsultaRecomendacaoOk>
13         </dataEstatico.ValorInteresse>
14         <dataEstatico.ValorInteresse>
15           <Valor>C#</Valor>
16           <Qualificador>1</Qualificador>
17           <ConsultaRecomendacaoOk>true</ConsultaRecomendacaoOk>
18         </dataEstatico.ValorInteresse>
19         <dataEstatico.ValorInteresse>
20           <Valor>C</Valor>
21           <Qualificador>1</Qualificador>
22           <ConsultaRecomendacaoOk>true</ConsultaRecomendacaoOk>
23         </dataEstatico.ValorInteresse>
24       </ValoresInteresse>
25     </dataEstatico.Interesse>
26   </ListaInteresses>
27 </dataEstatico.InteressesComunidade>
28
```

Algorithm 13: Xml de configuração do agente *Com02*

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <dataEstatico.InteressesComunidade>
4    <ListaInteresses>
5      <dataEstatico.Interesse>
6        <IdOntologia>PROGRAMACAO</IdOntologia>
7        <ValorPesoOntologia>1</ValorPesoOntologia>
8        <ValoresInteresse>
9          <dataEstatico.ValorInteresse>
10           <Valor>JAVA</Valor>
11           <Qualificador>1</Qualificador>
12           <ConsultaRecomendacaoOk>true</ConsultaRecomendacaoOk>
13         </dataEstatico.ValorInteresse>
14         <dataEstatico.ValorInteresse>
15           <Valor>C#</Valor>
16           <Qualificador>1</Qualificador>
17           <ConsultaRecomendacaoOk>true</ConsultaRecomendacaoOk>
18         </dataEstatico.ValorInteresse>
19       </ValoresInteresse>
20     </dataEstatico.Interesse>
21   </ListaInteresses>
22 </dataEstatico.InteressesComunidade>
23
```

Algorithm 14: Xml de configuração do agente *Com03*

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <dataEstatico.InteressesComunidade>
4    <ListaInteresses>
5      <dataEstatico.Interesse>
6        <IdOntologia>PROGRAMACAO</IdOntologia>
7        <ValorPesoOntologia>1</ValorPesoOntologia>
8        <ValoresInteresse>
9          <dataEstatico.ValorInteresse>
10           <Valor>C</Valor>
11           <Qualificador>1</Qualificador>
12           <ConsultaRecomendacaoOk>true</ConsultaRecomendacaoOk>
13         </dataEstatico.ValorInteresse>
14         <dataEstatico.ValorInteresse>
15           <Valor>PHP</Valor>
16           <Qualificador>1</Qualificador>
17           <ConsultaRecomendacaoOk>true</ConsultaRecomendacaoOk>
18         </dataEstatico.ValorInteresse>
19       </ValoresInteresse>
20     </dataEstatico.Interesse>
21   </ListaInteresses>
22 </dataEstatico.InteressesComunidade>
23
```

Algorithm 15: Xml de configuração do agente *Ag01*

```
<?xml version="1.0" encoding="utf-8"?>
<dataDinamico.BaseCrenca>
  <ConhecimentoAgenteBD>
    <listaConhecimentos>
      <dataDinamico.Conhecimento>
        <IdOntologia>C#</IdOntologia>
        <PesoNaOntologia>1</PesoNaOntologia>
        <listaValoresParaEsseNodoOntologia>
          <dataDinamico.Valor>
            <DescricaoValor>BUTTON</DescricaoValor>
            <QuantidadeRepeticao>10</QuantidadeRepeticao>
          </dataDinamico.Valor>
        </listaValoresParaEsseNodoOntologia>
      </dataDinamico.Conhecimento>
      <dataDinamico.Conhecimento>
        <IdOntologia>JAVA</IdOntologia>
        <PesoNaOntologia>1</PesoNaOntologia>
        <listaValoresParaEsseNodoOntologia>
          <dataDinamico.Valor>
            <DescricaoValor>BUTTON</DescricaoValor>
            <QuantidadeRepeticao>100</QuantidadeRepeticao>
          </dataDinamico.Valor>
          <dataDinamico.Valor>
            <DescricaoValor>LABEL</DescricaoValor>
            <QuantidadeRepeticao>10</QuantidadeRepeticao>
          </dataDinamico.Valor>
          <dataDinamico.Valor>
            <DescricaoValor>TABELA</DescricaoValor>
            <QuantidadeRepeticao>1</QuantidadeRepeticao>
          </dataDinamico.Valor>
        </listaValoresParaEsseNodoOntologia>
      </dataDinamico.Conhecimento>
      <dataDinamico.Conhecimento>
        <IdOntologia>PHP</IdOntologia>
        <PesoNaOntologia>1</PesoNaOntologia>
        <listaValoresParaEsseNodoOntologia>
          <dataDinamico.Valor>
            <DescricaoValor>PICTURE</DescricaoValor>
            <QuantidadeRepeticao>1</QuantidadeRepeticao>
          </dataDinamico.Valor>
        </listaValoresParaEsseNodoOntologia>
      </dataDinamico.Conhecimento>
      <dataDinamico.Conhecimento>
        <IdOntologia>C</IdOntologia>
        <PesoNaOntologia>1</PesoNaOntologia>
        <listaValoresParaEsseNodoOntologia>
          <dataDinamico.Valor>
            <DescricaoValor>COMANDOS</DescricaoValor>
            <QuantidadeRepeticao>1</QuantidadeRepeticao>
          </dataDinamico.Valor>
        </listaValoresParaEsseNodoOntologia>
      </dataDinamico.Conhecimento>
    </listaConhecimentos>
  </ConhecimentoAgenteBD>
  <InteressesUsuarioBD>
    <ListaInteresses>
      <dataEstatico.Interesse>
        <IdOntologia>PROGRAMACAO</IdOntologia>
        <ValorPesoOntologia>1</ValorPesoOntologia>
        <ValoresInteresse>
          <dataEstatico.ValorInteresse>
            <Valor>C#</Valor>
            <Qualificador>1</Qualificador>
          </dataEstatico.ValorInteresse>
          <dataEstatico.ValorInteresse>
            <Valor>JAVA</Valor>
            <Qualificador>1</Qualificador>
          </dataEstatico.ValorInteresse>
        </ValoresInteresse>
      </dataEstatico.Interesse>
    </ListaInteresses>
  </InteressesUsuarioBD>
</dataDinamico.BaseCrenca>
```

Algorithm 16: Xml de configuração do agente *Ag02*

```
<?xml version="1.0" encoding="utf-8"?>
<dataDinamico.BaseCrenca>
  <ConhecimentoAgenteBD>
    <listaConhecimentos>
      <dataDinamico.Conhecimento>
        <IdOntologia>JAVA</IdOntologia>
        <PesoNaOntologia>1</PesoNaOntologia>
        <listaValoresParaEsseNodoOntologia>
          <dataDinamico.Valor>
            <DescricaoValor>BUTTON</DescricaoValor>
            <QuantidadeRepeticao>5</QuantidadeRepeticao>
          </dataDinamico.Valor>
        </listaValoresParaEsseNodoOntologia>
      </dataDinamico.Conhecimento>
      <dataDinamico.Conhecimento>
        <IdOntologia>C#</IdOntologia>
        <PesoNaOntologia>1</PesoNaOntologia>
        <listaValoresParaEsseNodoOntologia>
          <dataDinamico.Valor>
            <DescricaoValor>BUTTON</DescricaoValor>
            <QuantidadeRepeticao>50</QuantidadeRepeticao>
          </dataDinamico.Valor>
          <dataDinamico.Valor>
            <DescricaoValor>TEXTBOX</DescricaoValor>
            <QuantidadeRepeticao>30</QuantidadeRepeticao>
          </dataDinamico.Valor>
        </listaValoresParaEsseNodoOntologia>
      </dataDinamico.Conhecimento>
    </listaConhecimentos>
  </ConhecimentoAgenteBD>
  <InteressesUsuarioBD>
    <ListaInteresses>
      <dataEstatico.Interesse>
        <IdOntologia>PROGRAMACAO</IdOntologia>
        <ValorPesoOntologia>1</ValorPesoOntologia>
        <ValoresInteresse>
          <dataEstatico.ValorInteresse>
            <Valor>C</Valor>
            <Qualificador>1</Qualificador>
          </dataEstatico.ValorInteresse>
          <dataEstatico.ValorInteresse>
            <Valor>C#</Valor>
            <Qualificador>1</Qualificador>
          </dataEstatico.ValorInteresse>
        </ValoresInteresse>
      </dataEstatico.Interesse>
    </ListaInteresses>
  </InteressesUsuarioBD>
</dataDinamico.BaseCrenca>
```

Algorithm 17: Xml de configuração do agente *Ag03*

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <dataDinamico.BaseCrenca>
4    <ConhecimentoAgenteBD>
5      <listaConhecimentos>
6        <dataDinamico.Conhecimento>
7          <IdOntologia>C#</IdOntologia>
8          <PesoNaOntologia>1</PesoNaOntologia>
9          <listaValoresParaEsseNodoOntologia>
10           <dataDinamico.Valor>
11             <DescricaoValor>BUTTON</DescricaoValor>
12             <QuantidadeRepeticao>100</QuantidadeRepeticao>
13           </dataDinamico.Valor>
14         </listaValoresParaEsseNodoOntologia>
15       </dataDinamico.Conhecimento>
16     </listaConhecimentos>
17   </ConhecimentoAgenteBD>
18   <InteressesUsuarioBD>
19     <ListaInteresses>
20       <dataEstatico.Interesse>
21         <IdOntologia>PROGRAMACAO</IdOntologia>
22         <ValorPesoOntologia>1</ValorPesoOntologia>
23         <ValoresInteresse>
24           <dataEstatico.ValorInteresse>
25             <Valor>C#</Valor>
26             <Qualificador>1</Qualificador>
27           </dataEstatico.ValorInteresse>
28           <dataEstatico.ValorInteresse>
29             <Valor>C</Valor>
30             <Qualificador>1</Qualificador>
31           </dataEstatico.ValorInteresse>
32         </ValoresInteresse>
33       </dataEstatico.Interesse>
34     </ListaInteresses>
35   </InteressesUsuarioBD>
36 </dataDinamico.BaseCrenca>
37
```

Algorithm 18: Xml de configuração do agente *Ag04*

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <dataDinamico.BaseCrenca>
4    <ConhecimentoAgenteBD>
5      <listaConhecimentos>
6        <dataDinamico.Conhecimento>
7          <IdOntologia>JAVA</IdOntologia>
8          <PesoNaOntologia>1</PesoNaOntologia>
9          <listaValoresParaEsseNodoOntologia>
10           <dataDinamico.Valor>
11             <DescricaoValor>LABEL</DescricaoValor>
12             <QuantidadeRepeticao>60</QuantidadeRepeticao>
13           </dataDinamico.Valor>
14           <dataDinamico.Valor>
15             <DescricaoValor>BUTTON</DescricaoValor>
16             <QuantidadeRepeticao>10</QuantidadeRepeticao>
17           </dataDinamico.Valor>
18         </listaValoresParaEsseNodoOntologia>
19       </dataDinamico.Conhecimento>
20       <dataDinamico.Conhecimento>
21         <IdOntologia>C#</IdOntologia>
22         <PesoNaOntologia>1</PesoNaOntologia>
23         <listaValoresParaEsseNodoOntologia>
24           <dataDinamico.Valor>
25             <DescricaoValor>BUTTON</DescricaoValor>
26             <QuantidadeRepeticao>50</QuantidadeRepeticao>
27           </dataDinamico.Valor>
28         </listaValoresParaEsseNodoOntologia>
29       </dataDinamico.Conhecimento>
30     </listaConhecimentos>
31   </ConhecimentoAgenteBD>
32   <InteressesUsuarioBD>
33     <ListaInteresses>
34       <dataEstatico.Interesse>
35         <IdOntologia>PROGRAMACAO</IdOntologia>
36         <ValorPesoOntologia>1</ValorPesoOntologia>
37         <ValoresInteresse>
38           <dataEstatico.ValorInteresse>
39             <Valor>JAVA</Valor>
40             <Qualificador>1</Qualificador>
41           </dataEstatico.ValorInteresse>
42           <dataEstatico.ValorInteresse>
43             <Valor>PHP</Valor>
44             <Qualificador>1</Qualificador>
45           </dataEstatico.ValorInteresse>
46         </ValoresInteresse>
47       </dataEstatico.Interesse>
48     </ListaInteresses>
49   </InteressesUsuarioBD>
50 </dataDinamico.BaseCrenca>
51
```

Algorithm 19: Xml de configuração do agente *Ag05*

```
1
2  <?xml version="1.0" encoding="utf-8"?>
3  <dataDinamico.BaseCrenca>
4    <ConhecimentoAgenteBD>
5      <listaConhecimentos>
6        <dataDinamico.Conhecimento>
7          <IdOntologia>PHP</IdOntologia>
8          <PesoNaOntologia>1</PesoNaOntologia>
9          <listaValoresParaEsseNodoOntologia>
10           <dataDinamico.Valor>
11             <DescricaoValor>BUTTON</DescricaoValor>
12             <QuantidadeRepeticao>100</QuantidadeRepeticao>
13           </dataDinamico.Valor>
14         </listaValoresParaEsseNodoOntologia>
15       </dataDinamico.Conhecimento>
16     </listaConhecimentos>
17   </ConhecimentoAgenteBD>
18   <InteressesUsuarioBD>
19     <ListaInteresses>
20       <dataEstatico.Interesse>
21         <IdOntologia>PROGRAMACAO</IdOntologia>
22         <ValorPesoOntologia>1</ValorPesoOntologia>
23         <ValoresInteresse>
24           <dataEstatico.ValorInteresse>
25             <Valor>PHP</Valor>
26             <Qualificador>1</Qualificador>
27           </dataEstatico.ValorInteresse>
28         </ValoresInteresse>
29       </dataEstatico.Interesse>
30     </ListaInteresses>
31   </InteressesUsuarioBD>
32 </dataDinamico.BaseCrenca>
33
```