UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

2005.2

APLICAÇÃO DE CONTEXTO AO COWS
UMA FERRAMENTA DE ESCRITA COLABORATIVA

Aluno: Diego Reynaldo Lira Llamoca Zárate (drllz@cin.ufpe.br)

Orientadora: Profa. Ana Carolina Salgado (acs@cin.ufpe.br)

Janeiro de 2006

Recife - Brasil

Aos meus pais David e Maria.
A minha irmã Diana.
A todos os meus amigos

Agradecimentos

A minha orientadora Carol, pelo apoio durante a confecção do trabalho e por ser para mim um exemplo de professora, dentro e fora de sala de aula.

A minha co-orientadora, Vaninha, que idealizou esse trabalho. Pela imensa ajuda que me deu e as todas as coisas que me ensinou desde que a conheci.

A meus pais David e Maria pelo amor, incentivo e por terem me agüentado durante toda a minha vida.

A Diana e Erica pela ajuda que me deram, não me esqueci de vocês.

Aos amigos Ivan, Augusto, Pedro, Gilberto, Eudes. Pelo incentivo e apoio.

Ao Colégio de Aplicação, Centro de Informática e Universidade Federal de Pernambuco pela formação.

A todos os outros que, de alguma forma, contribuíram para o desenvolvimento deste trabalho, meus sinceros agradecimentos.

RESUMO: Desde a criação da idéia da web semântica alguns pesquisadores vem propondo modelos de criação de sistemas baseados em ontologias. Já foram propostas várias ontologias em várias áreas, esse trabalho se baseia em uma ontologia para sistemas cientes de contexto, para com a dessa ontologia adicionar a capacidade de ciência de contexto a uma sistema de escrita colaborativa chamado CoWS. Basicamente foi inserido ao CoWS a capacidade de gerenciar configurações do usuário em um perfil (que fica armazenado na ontologia) e, de acordo com o nível de detalhe que o usuário preencheu no seu perfil, o sistema apresenta uma janela destacando os textos que o usuário possivelmente tem interesse (baseado no seu perfil). Para fazer isso, existem regras que foram criadas para a ontologia, as quais são validadas e executadas usando uma máquina de inferência embutida em um framework web semântico chamado Jena, que lê e trata tanto a ontologia quando as regras definidas sobre essa ontologia. A criação de um sistema ciente de contexto baseado em ontologias não é algo simples de ser feito, mas se apresenta como uma opção promissora pelas possibilidades de aproveitamento, expansão de sistemas baseados em ontologias, o compartilhamento entre o conhecimento contextual entre máquinas e homens e interoperabilidade.

Palavras-chave: percepção, contexto, CSCW, *groupware*, computação ciente de contexto.

Abstract: Since the creation of the web Semantics ideas, some researchers proposed models for creation of ontology-based systems. There are already some proposed ontologies in a handful areas, this work is based on a Ontology for context-aware systems, this ontology is used for adding the feature of context-awareness in a collaborative application called CoWS. Basically in CoWS was inserted the capacity of manage the users configurations in a profile (stored in the ontology) and based on the level of detail provided by the user on the profile, the system show a window highlighting the texts that the user probably has most interest (profile based). To manage that, some rules were created for the ontology, which are validated and run using a web semantic framework called Jena. Jena read and manage both the ontology and its rules. The creation of a ontology-based system its no easy task, but show some promising possibilities: easy evolution and expansion, knowledge sharing between human and machine and interoperability.

Keywords: awareness, context, CSCW, groupware, context-awareness

Índice

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

1 Introdução

Um aplicativo para edição colaborativa é parte do campo de pesquisa de CSCW (*Computer Supported Cooperative Work*, em português Trabalho Cooperativo Apoiado por Computador), mais especificamente como uma ferramenta de *groupware*, termo que se refere a sistemas computacionais que auxiliam grupos de pessoas engajadas em uma tarefa ou objetivo comum e que fornecem uma interface para um ambiente compartilhado (ELLIS et al., 1991). Esses aplicativos precisam lidar com várias dificuldades que são inerentes ao trabalho em grupo. Por exemplo, como ajudar pessoas que têm horários que não são conciliáveis a criar um documento em conjunto? A questão não se restringe apenas a horários difíceis de conciliar, mas também a locais (pessoas de diferentes países). Uma idéia para lidar com as dificuldades mencionadas e até mesmo aumentar a eficácia desse tipo de ferramenta poderia ser a implantação de um mecanismo de percepção de contexto (ROSA et al., 2005). Percepção é a capacidade de cada usuário estar constantemente informado das atividades realizadas pelos seus colegas.

Contexto é uma área relativamente nova da Ciência da Computação. A discussão sobre o significado de contexto computacional e como fazer aplicações que detectem e lidem com a idéia de contexto começou apenas em 1994 (SCHILIT, 1994). Desde então, vêm surgindo várias definições e estudos sobre como aplicar contexto em diversas áreas da computação, principalmente no domínio de aplicações móveis e ubíquas (CHEN e KOTZ, 2000; DEY e ABOWD, 2000; GU et al., 2004a), mas também em sistemas colaborativos (ALARCON et al., 2005; ROSA et al., 2005). Um aplicativo ciente de contexto tem a capacidade de, com base em informações fornecidas pelo usuário ou capturadas pelo próprio aplicativo, mudar sua interface ou apresentar alguma informação que vá ajudar o usuário em sua situação atual.

Como o objetivo principal dos sistemas colaborativos é promover a interação e a colaboração entre os seus usuários, a adição de mecanismos de contexto poderá ser muito útil no apoio ao trabalho em grupo, pois permite que o aplicativo forneça serviços e informações relevantes a cada um dos membros do

grupo durante a execução de suas tarefas colaborativas (VIEIRA et al., 2005a). Isso vai evitar a sobrecarga de informações e trabalhos repetitivos e redundantes.

O CoWS (*Collaborative Writing through Shared Spaces*) (VIEIRA et al., 2005c) é uma ferramenta que tem por objetivo apoiar a criação colaborativa de documentos textuais, focando no desenvolvimento móvel e com suporte tanto à escrita síncrona (*online*) quanto assíncrona (*offline*). O CoWS utiliza um *middleware* baseado em espaços de tuplas, chamado Lime[1] (*Linda in Mobile Environments*), que serve para aliviar as complexidades inerentes de comunicação (integração) nesse tipo de aplicativo. Atualmente, não existe nada relacionado à aplicação de contexto no CoWS e essa é a idéia deste trabalho: dar os primeiros passos em direção à inserção das funcionalidades de captura, raciocínio e visualização de contexto nessa ferramenta, é esperado que um sistema de escrita colaborativa com ciência de contexto se torne uma ferramenta mais eficiente para seus usuários ao mostrar para eles informações que o ajudem a escrever o seu texto.

1.1 Objetivos

Estender o CoWS para torná-lo ciente de contexto. Para isso, será usado o modelo de ontologia de contexto definido em(VIEIRA et al., 2005a), será adicionada ao sistema a capacidade de inferir novos dados de contexto, definindo para isso algumas regras de inferência com base em configurações do perfil do usuário e do texto sendo editado. Esse contexto será utilizado para destacar parágrafos mais relevantes para o usuário. Assim, espera-se que seja obtido um exemplo prático do funcionamento de um aplicativo ciente de contexto baseado em um modelo ontológico e usando regras de inferência.

1.2 Organização da monografia

O trabalho está organizado em 6 (seis) capítulos:

No capítulo 2 (dois) o leitor irá conhecer um pouco sobre as definições de CSCW, *groupware*, escrita colaborativa, percepção e contexto computacional.

[1] Lime está disponível em http://lime.sourceforge.net/

O capítulo 3 (três) analisa trabalhos similares ao desenvolvido, em relação as capacidades e funções de alguns sistemas de escrita colaborativas. com relação às propostas de ontologia, abrangência da aplicação e um comparativo com o modelo proposto neste trabalho.

O capítulo 4 (quatro) explica o sistema CoWS, desde o surgimento da idéia, o problema que ele se propõe a solucionar e a ontologia que está sendo usada neste trabalho para validar a extensão do CoWS.

No capítulo 5 (cinco) é detalhada a nossa proposta de arquitetura para a nova versão ciente de contexto do CoWS, além de explicar a implementação, os requisitos, os problemas que surgiram e os resultados obtidos..

O capítulo 6 (seis) apresenta as conclusões do trabalho desenvolvido, assim como algumas projeções futuras.

2 Revisão da Literatura

Este capitulo apresenta os conceitos básicos que formaram a idéia inicial para a elaboração da versão atual do CoWS e, consequentemente, deste projeto, além de novos conceitos usados neste trabalho.

2.1 CSCW

Computer Supported Cooperative Work (Trabalho Cooperativo Apoiado por Computador) é a área de pesquisa que estuda o uso de tecnologias de computação e comunicação para dar suporte a atividades em grupo. Essa área procura responder a perguntas do tipo “Como as pessoas interagem para criar um trabalho colaborativo?” ou “Como a tecnologia pode facilitar e melhorar essa interação?” (ELLIS e WAINER, 1991). Associado as pesquisas de CSCW surgiram *groupwares* (classe de aplicativos que criam ambientes virtuais compartilhados, maiores detalhes na seção 2.2), novos paradigmas, novos tipos de sistemas, e novas maneiras de trabalhar (ELLIS e WAINER, 1991). Junto com essas oportunidades surgiram também novos problemas e desafios, para criar aplicativos de *groupware* cada vez melhores baseados nos preceitos de CSCW.

As metodologias de pesquisa utilizadas na área de CSCW incluem: estudo de campo, experimentos em laboratório, estudos etnográficos, simulações e modelagem conceitual. Essa área já é bem pesquisada e já foram realizados muitos estudos com várias técnicas distintas. Os resultados desses trabalhos serviram para melhorar tanto a interação em reuniões cara-a-cara (síncronas) quanto em interações assíncronas (tipo o CoWS em que os usuários não interagem no mesmo instante de tempo) (ELLIS e WAINER, 1991).

Na área de CSCW, o suporte tecnológico vem mudando e evoluindo em uma velocidade muito maior do que o entendimento da interação humana (ELLIS e WAINER, 1991). Segundo Ellis ainda é necessário um nível muito mais profundo de conhecimento dos fatores sociais e organizacionais e suas interações com a tecnologia. Por isso, um componente importante do estudo de CSCW são as teorias, *frameworks* (como o Lime) e modelos matemáticos.

CSCW ainda inclui os estudos teóricos de modelos de times, organizações e sistemas sociais. Ellis (ELLIS e WAINER, 1991)também considera que esses

estudos dão suporte à análise, previsão e projeto de estruturas sociais, e devem levar em conta as características dos participantes, possibilidades de comunicação, objetivos, relacionamentos e mecanismos de incentivo à interação.

Na construção dessas teorias, os estudos de CSCW se espalham por diversas áreas distintas como, psicologia social, projeto organizacional, economia, ciência da computação e administração (ELLIS e WAINER, 1991). Enquanto as tecnologias da informação vão buscar os fatores de mais baixo nível, tais como as possibilidades de comunicação, os modelos teóricos provêm um meio de avaliar os efeitos de diferentes tipos de projeto, servindo como guia para unir a tecnologia aos sistemas sociais, para criar aplicativos cada vez melhores.

2.2 Groupware

A tecnologia de *groupware* apóia o trabalho em grupo através da criação de ambientes virtuais compartilhados, nos quais os participantes do grupo buscam a cooperação/colaboração (estou considerando os dois termos com o mesmo significado embora alguns autores discordem (DILLENBOURG et al., 1996)). Existem muitos tipos de *groupware*, dentre os mais conhecidos: programas de vídeo conferência (usuários interagindo ao mesmo tempo, mas em locais diferentes); sistemas de suporte à decisão em grupo (GDSS - *Group Decision Support Systems*), que são sistemas usados em reuniões com suporte a apresentações, votação e *brainstorming*, onde tipicamente todos os usuários estão no mesmo lugar na mesma hora; e sistemas de autoria colaborativa, com possibilidade dos usuários usarem o sistema tanto em locais quanto em horas distintas, como no caso do CoWS.

Um tipo de ambiente que pode ser considerado uma categoria de *groupware* que está se tornando bastante popular são os ambientes de memória compartilhada. Esses ambientes têm como propósito facilitar a troca de idéias, conhecimentos a até arquivos de um grupo interessado em um determinado assunto. Por exemplo, existe um grupo de discussão sobre o *framework* Jena. Quando um membro do grupo de discussão do Jena tem uma dúvida, ele envia uma pergunta para um servidor. Esse servidor é responsável por guardar a mensagem para que posteriormente qualquer membro do referido grupo possa lê-la (com isso é possível que membros que entraram no grupo depois do envio

dessa mensagem possam ler e aprender com ela), nenhuma mensagem ou arquivo é enviado para um membro em específico, fica tudo armazenado no servidor. Grupos de discussão desse tipo como o Yahoo groups e, recentemente, o Google groups, são os mais populares.

Todos esses exemplos mostram uma ampla variedade de aplicações, mas todos têm em comum o foco em quatro aspectos principais, inter-relacionados (Fig. 1): a **comunicação** entre os membros do grupo, a **coordenação** de suas atividades, o armazenamento e recuperação do conhecimento comum através da **memória do grupo** e a **percepção** do contexto do trabalho do grupo, por seus membros (ARAÚJO, 2000).

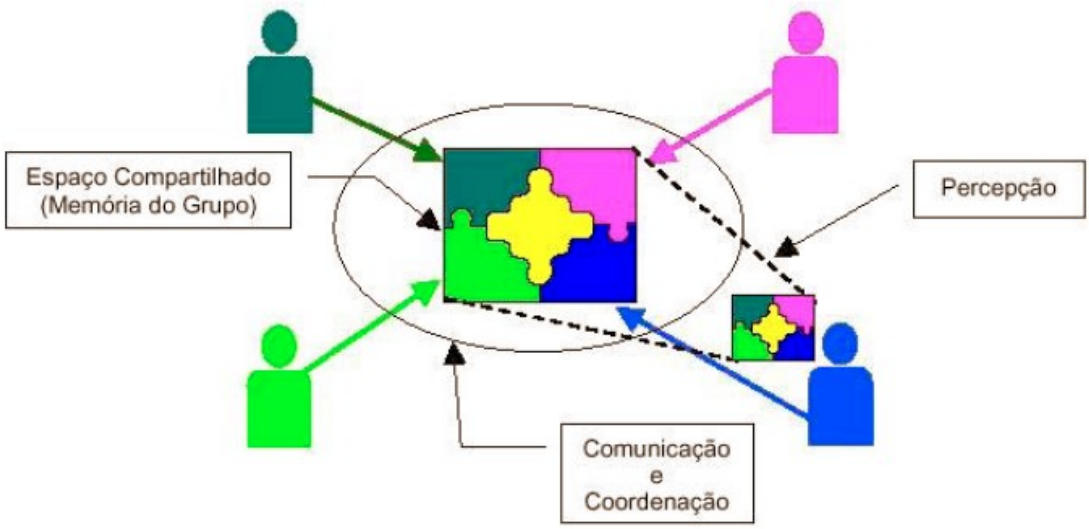

Fig. 1. Esquema geral dos aspectos do trabalho em grupo (ARAÚJO, 2000)

Segundo Rosa *et al* (ROSA et al., 2005), um dos aspectos mais importantes relacionados ao apoio à cooperação é o contexto onde o grupo interage. Porém, diversos autores apontam que as ferramentas de *groupware* ainda não dão a devida atenção ao uso de contexto (ALARCON et al., 2005; BRÉZILLON et al., 2004; VIEIRA et al., 2005a).

2.2.1 Escrita colaborativa

A área de CSCW que pesquisa como as ferramentas de *groupware* podem auxiliar dois ou mais autores a conjuntamente produzir um único documento é designada como escrita ou autoria colaborativa (*collaborative writing*) (SAPSOMBOON et al., 1997).

A tarefa de criação de um texto é bem complexa (POSNER e E BAECKER, 1992) e se torna mais complicada ainda quando desenvolvida em grupo. Devido a questões como facilidades de edição, controle de versões, documentação, agenciamento de esforços necessários para a sincronização das atividades, e necessidade de acompanhamento do processo criativo, a atividade de escrita coletiva é uma forte candidata a ter um suporte da informática através dos sistemas de autoria coletiva.

O principal objetivo dessas ferramentas é o de incentivar os participantes a trabalhar em grupo, num documento comum, respeitando as características individuais (GONÇALVES e PADILHA, 2003). As ferramentas de edição colaborativa devem possuir funcionalidades e características que facilitem o desenvolvimento de textos entre autores que estejam interagindo de forma colaborativa. Dentre as características técnicas que essas ferramentas devem atender destacam-se: o controle de concorrência, o suporte à percepção, o tipo de arquitetura da ferramenta (distribuída ou centralizada), e o modo de interação entre os usuários, síncrono (no mesmo instante de tempo) ou assíncrono (em instantes diferentes).

O controle de concorrência serve para evitar ou diminuir a ocorrência de conflitos entre os usuários, uma vez que os mesmos realizam operações simultâneas em documentos compartilhados (GONÇALVES e PADILHA, 2003).

A percepção tem como objetivo propagar as alterações feitas no texto por um autor para todos os outros autores de forma eficiente e que não atrapalhe os usuários na realização de suas tarefas.

Há ferramentas que possibilitam que os usuários trabalhem assincronamente, em modo *offline*. Nesse caso, a percepção não é imediata, pois a contribuição do usuário só será visualizada no momento em que houver a conexão e o compartilhamento de dados com os demais usuários. Esse problema não existe em aplicativos apenas síncronos, uma vez que todas as ações dos usuários são imediatamente propagadas para os demais.

Por fim, no projeto de uma ferramenta colaborativa pode-se optar por uma arquitetura distribuída ou centralizada. Na primeira, o armazenamento dos documentos ocorre em diversos *hosts*. Tipicamente o servidor armazena o documento e os clientes possuem uma cópia desse documento, ficando a cargo do controlador de versões verificar as atualizações existentes. Com isso, os clientes podem trabalhar independentemente do funcionamento da rede. No entanto, na

arquitetura centralizada os documentos são armazenados em um servidor central que fica responsável por manter os arquivos compartilhados consistentes. Uma desvantagem dessa abordagem é que se o servidor não funcionar, os clientes não poderão recuperar versões mais recentes nem compartilhar suas versões.

Vários sistemas de autoria colaborativa mediados por computador têm sido desenvolvidos. Cada um destes sistemas tem um enfoque particular. Uns se preocupam com o desempenho da rede, outros com o processo de autoria, outros ainda com a estruturação dos documentos, hipermídia, etc. Os primeiros sistemas de autoria coletiva utilizavam redes de computadores, LANs ou WANs, para proporcionar a interação entre os usuários (AXT et al., 2005). Isso limitou a atuação desses sistemas restringindo seu uso a usuários da mesma plataforma empregada. Com as facilidades de comunicação e interação, nos dias atuais, proporcionadas pela Internet, interligando os mais distintos pontos do nosso planeta, diversas ferramentas de cunho colaborativo/cooperativo têm sido desenvolvidas. A Internet possibilita uma independência de plataforma e uma forma padrão de acesso (através dos navegadores ou *browsers*), apesar de apresentar algumas dificuldades para trabalhos síncronos.

2.3 Percepção

Percepção (ou do inglês, *awareness*) é um conceito já bastante usado em pesquisas de CSCW. A definição mais referenciada é a de Dourish e Belloti (DOURISH e BELLOTTI, 1992) que diz que percepção “é a compreensão das atividades dos outros, que trazem contexto para a sua própria atividade”. Em decorrência disso, pode-se observar que a percepção é fundamental para trabalhos feitos em grupos cooperativos em ambientes computacionais. É importante conhecer e compreender as atividades realizadas pelos colegas de trabalho, sendo muitas vezes uma necessidade que o trabalho individual seja divulgado para os outros. Em fóruns de discussão, por exemplo, uma característica interessante é que os usuários vejam as mensagens novas (as que foram enviadas depois da sua última visita ao fórum) em destaque em relação às antigas. Isso é percepção.

Percepção também pode ser conceituada como o ato de se contextualizar informações, de mostrar o contexto onde o usuário está trabalhando no momento (ROSA et al., 2005). Cada membro pode perceber a mesma informação de

contexto de maneira diferente, pois o ato de perceber está associado com a cognição do indivíduo. Esse conceito se parece muito com o conceito de contexto, que vai ser detalhado na próxima seção, a ponto de alguns autores considerarem que são indissociáveis.

2.4 Contexto Computacional

Nas iterações do dia-a-dia, as pessoas têm uma idéia do que significa contexto, mas contexto aplicado à computação ainda é um campo em aberto sem uma definição de consenso.

2.4.1 Definições existentes sobre contexto

Segundo Dey *et al*., o primeiro trabalho que cita o termo computação ciente de contexto é o de Schilit (SCHILIT e THEIMER, 1994), os quais definem contexto como a localidade, identidade de pessoas e objetos perto, e mudanças nesses objetos. Schilit (SCHILIT et al., 1994) defende que os aspectos mais importantes para contexto são: “onde você está”, “com quem você está” e “que recursos estão perto”. De posse desses dados, contexto é considerado como as mudanças constantes no ambiente de execução. Eles incluem ainda as seguintes características do ambiente:

- Recursos computacionais: processadores disponíveis, dispositivos de entrada e saída, capacidade da rede, conectividade e custos computacionais;
- Ambiente do usuário: localização, pessoas próximas, situação social;
- Ambiente físico: níveis de luz e ruído.

Brézillon (BRÉZILLON et al., 2004) classificam informações contextuais em duas categorias: conhecimento externo e conhecimento contextual. **Conhecimento externo** é a parte de contexto que não é relevante para o foco atual, o **conhecimento contextual** tem ligação com o foco atual, mas não é considerado ainda pela aplicação. Dependendo do foco do usuário uma parte do conhecimento contextual é extraída, montada e estruturada em um **contexto proceduralizado**. Para que todo esse processo seja realizado é necessário um *framework* que explique e anteveja os resultados de uma decisão ou ação do usuário. No caso, esse framework poderia ser uma ontologia junto com uma máquina de inferência como vai ser usado nesse trabalho.

2.4.2 Conceito de Contexto Adotado

Neste trabalho, a base conceitual para contexto será principalmente as definições propostas por Dey e Abowd (DEY e ABOWD, 2000) e por Brézillon (BRÉZILLON et al., 2004) que são as mais referenciadas na literatura.

Seguindo a definição de Dey e Abowd (DEY e ABOWD, 2000) fica mais fácil a enumeração de contexto para o desenvolvedor de uma aplicação diversa. Se alguma informação pode ser usada para caracterizar a situação do participante em uma interação, então essa informação é contexto. Exemplificando, na construção de uma tabela de pesos no Excel, as entidades são os usuários e a aplicação (Excel). Considerando a presença de outras pessoas perto do usuário e a localização do mesmo, qual dessas informações pode ser considerada contexto para essa aplicação? A presença de outras pessoas não influi em nada para o usuário do Excel, logo não é contexto nesse caso. Porém, a localização influi na unidade de medida, pois se o usuário for brasileiro ele usará quilograma como unidade e se o usuário for americano ele, provavelmente, irá usar outra unidade de medida. Conclui-se que a localização do usuário é contexto, pois pode ser utilizada para definir a situação do usuário.

Um erro que alguns autores cometem, na opinião de Dey e Abowd (DEY e ABOWD, 2000), é que os demais autores consideram que contexto é somente uma informação implícita (que foi deduzida a partir de outra), enquanto que para eles tanto dados explicitamente gerados pelo usuário quanto dados automaticamente deduzidos são ambos considerados formas de contexto. Se a identidade de um usuário foi detectada de algum modo automático, ou simplesmente o usuário quando se conectou ao sistema informou sua identidade, pouco importa, pois essa informação é considerada contexto. Pesquisas baseadas em obtenção de informações implícitas estão mais em foco porque essa é uma área menos explorada (obtenção de informações da interação homem-máquina) (DEY e ABOWD, 2000).

2.4.3 Categorias de contexto

Uma categorização dos tipos de contexto ajuda os desenvolvedores de aplicativos a descobrir os tipos de dados mais interessantes para seus aplicativos.

Schilit *et al*. (SCHILIT et al., 1994) lista “onde você está”, “com quem você está”, “que recursos estão disponíveis” como aspectos importantes de contexto.

Aplicativos cientes de contexto procuram por quem é, onde está, quando foi e o que foi (resumidamente o que o usuário vem fazendo) e usa essa informação para determinar o porquê da situação estar ocorrendo. Um aplicativo na verdade não determina porque a situação está ocorrendo, mas o desenvolvedor sim. O desenvolvedor usa o contexto recebido para determinar porque a situação está ocorrendo e usa isso para codificar alguma ação no aplicativo. Por exemplo, em um guia turístico ciente de contexto, um usuário com um computador portátil se aproxima de um monumento histórico, o que faz com que o aplicativo mostre informações sobre aquele monumento. Nessa situação, o desenvolvedor codificou no aplicativo a capacidade de que quando um usuário se aproximar de um lugar particular (contexto recebido), significa que o usuário está interessado no monumento, e o aplicativo deve mostrar alguma informação relevante (ação).

Alguns tipos de contexto são considerados básicos ou primários. Normalmente, são considerados básicos: localização, identidade, atividade e tempo (WANG et al., 2004). Atividade por outro lado serve de resposta para a importante questão “o que está acontecendo na situação”. As categorias criadas por Schilit *et al*. (SCHILIT et al., 1994) (“onde você está”, “com quem você está”, “que objetos estão próximos a você”) só incluíam informação de localidade e identidade. Para caracterizar uma situação também são necessárias informações de atividade (“o que você está fazendo”) e tempo (“em que horário”).

Com base nesses tipos de contexto básicos além de responder às questões “quem”, “o que”, “quando” e “onde”, essas informações também podem funcionar como índices para outras fontes de informação contextual (DEY e ABOWD, 2000). Por exemplo, dada a identidade de uma pessoa, podem ser recuperadas muitas informações correlatas, como o telefone, e-mail, data de nascimento, lista de amigos e relacionamentos. Com a localização podem ser determinados que outros objetos ou pessoas estão perto da entidade (pessoa) e que atividades estão ocorrendo por perto. Com base nesses exemplos, percebemos que os dados de contexto primário podem ser usados como índice para achar contexto secundário (e-mail, por exemplo) para a mesma entidade ou até mesmo contexto primário de entidades relacionadas (outras pessoas no mesmo lugar, por exemplo) (DEY e ABOWD, 2000).

Nessa divisão de contexto primário e secundário, temos uma arquitetura em dois níveis (DEY e ABOWD, 2000). Os quatro tipos de contexto que foram destacados são do primeiro nível, todo o resto é do segundo nível. Os tipos de contexto secundários têm uma característica em comum: eles podem ser indexados pelo contexto primário porque eles são atributos da entidade visada. Por exemplo, o telefone do usuário é um dado de contexto secundário e pode ser obtido usando a identidade do usuário em uma lista telefônica. Em algumas situações, são necessárias múltiplas unidades de contexto primário para adquirir o contexto secundário num índice qualquer. Por exemplo, para informar a previsão do tempo um sistema ciente de contexto precisa saber tanto a localização do usuário quanto a data em que é desejada a previsão. Essa categorização ajuda os desenvolvedores na escolha do contexto em suas aplicações, na estruturação do contexto usado e na busca por contexto relevante. As quatro informações de contexto primárias indicam que tipos de informação são necessários para caracterizar uma situação e seus usos como índices provêem um meio para organizar e usar o contexto.

2.4.4 Modelos de representação de contexto

Tao Gu *et al (GU et al., 2004b)* classificam os modelos de contexto em três categorias:

- *Abordagem orientada à aplicação*: Os sistemas cientes de contexto usam um modelo e representam a informação contextual para aplicações específicas. Esses modelos são normalmente proprietários e muito restritos, sem formalismo e reconhecimento. O projeto HP's Cooltown (KINDBERG e BARTON, 2001) propõe um modelo para web, onde cada objeto (pessoa, lugar e coisa) tem uma descrição que pode ser recuperada buscando uma URL. O sistema Context Toolkit (DEY et al., 2001) envia dados de contexto de baixo nível (arquivos XML no formato de par nome-valor), adquiridos por meio de sensores;

- *Abordagem orientada a modelo*: essa categoria usa modelagens conceituais para representar contexto. Um modelo formal baseado no modelo ER foi proposto por vários projetos (HARTER et al., 2002; WU. et al., 2002). Até porque dados de contexto podem ser facilmente

manuseados em bancos de dados relacionais. O modelo de Henricksen *et al*. (HENRICKSEN et al., 2002) e seus atributos adicionais (incluindo características de tempo e classificação) usa tanto modelos ER quanto diagramas UML. Esse modelo depois foi reformulado para o *Object-Role Modeling* (ORM) (HENRICKSEN et al., 2003).

- *Abordagem orientada a ontologias*: Essa abordagem é mais focada na construção de uma ontologia contextual para um domínio específico (a área em que vai ser usado o sistema) com o objetivo de permitir o compartilhamento dos conhecimentos através de vários sistemas (sistemas distribuídos). O *Comprehensive Structured Context Profiles* (CSCP) (HELD et al., 2002) foi desenvolvido baseado em RDF para representar contexto por meio de sessões de perfil. Chen *et al (CHEN et al., 2005)* definiu uma ontologia de contexto baseada em OWL para dar suporte a agentes ubíquos no seu Context Broker Architecture (CoBrA) (CHEN et al., 2003). Ranganathan *et al*. (RANGANATHAN e CAMPBELL, 2003) desenvolveu um *middleware* para criar aplicações cientes de contexto com interoperabilidade semântica, no qual eles representam a ontologia contextual usando DAML+OIL (HORROCKS, 2002).

As conclusões que podem ser tiradas das três categorias acima são basicamente as seguintes: a abordagem orientada à aplicação é a mais fraca, pois carece tanto de uma base teórica (formalismo) quanto de um suporte a compartilhamento da base de conhecimento entre sistemas diferentes. A abordagem orientada a modelo tem base num formalismo mais rigoroso, mas não trata de questões de compartilhamento de conhecimento (pelo menos não pode ser feito de maneira fácil), tampouco de inferência de contexto. A abordagem ontológica por sua vez foca na criação de uma ontologia que explore a capacidade de inferência baseada em tecnologias de Web Semântica, porém as ontologias existentes deixam a desejar em generalização e não resolveram plenamente questões como classificação de contexto, dependência de contexto e qualidade do contexto que vai ser útil na inferência contextual. Neste trabalho vai ser usada a abordagem ontológica proposta por Vieira *et al* (VIEIRA et al., 2005a), a qual será detalhada no capítulo 5. Essa ontologia foi pensada como um padrão para sistemas colaborativos, como é o caso do CoWS, diferentemente da maioria das

outras ontologias propostas para sistemas cientes de contexto que almejavam principalmente computação ubíqua.

Essa teoria foi a base teórica para a versão inicial do CoWS

3 Trabalhos Relacionados

Neste capítulo vamos comentar sobre alguns projetos similares ao CoWS. Não conseguimos achar nenhum aplicativo que tivesse o foco de ser um editor colaborativo ciente de contexto, mas existem vários editores de escrita colaborativa.

3.1 Alguns sistemas de escrita colaborativa

Essa seção faz uma breve descrição dos sistemas de escrita colaboraivos pesquisados, nenhum deles tem configuração de preferências (o meio pelo qual o CoWS possui ciência de contexto) no nível da nova versão do CoWS.

- **Bloki** em www.bloki.com
  Beta, focado para criação de páginas web em grupo, pode ser usado também para escrita colaborativa. Usa a web como processador de texto.

- **REDUCE** em http://reduce.qpsf.edu.au/
  Esse sistema permite edição síncrona. Usa a web como processador.

- **CoWord** em http://reduce.qpsf.edu.au/coword/
  Mesmos criadores do REDUCE. Permite a edição síncrona de documentos no MS Word.

- **EquiText** em http://equitext.pgie.ufrgs.br/
  Usa a web como editor, foi feito por um grupo brasileiro.

- **Gobby** em http://gobby.0x539.de/
  Editor síncrono mais focado para criação de código em grupo, funciona em Windows, Mac, Linux e Unix.

- **LiveDrive and Collaboration Manager for Word** em http://www.chasseral.com/products/index.shtml

Plug-in comercial para o MS Word, permite que vários usuários do MS Word trabalhem em uma cópia central de um documento.

- **DocReview** em http://purl.oclc.org/DocReview/get
  Unix apenas, permite escrita de comentários em algum documento, usa a web para armazenar comentários, não modifica o arquivo original.

- **MoonEdit** em http://me.sphere.pl/indexen.htm
  Editor de texto multi-plataforma, permite que vários autores trabalhem ao mesmo tempo no mesmo documento pela web.

- **SubEthaEdit** em http://www.codingmonkeys.de/subethaedit/
  Para o Mac OS X. SubEthaEdit é focado na criação de código colaborativo mas pode funcionar como editor colaborativo em tempo real. Ferramenta *peer-to-peer*.

- **SynchroEdit** em http://www.synchroedit.com/
  Ainda em versão alfa, editor síncrono multi-plataforma, que usa a web como processador de texto.

- **SparrowWeb** em
  http://www.alphaavenue.com/details.php?tech=Sparrow%20Web
  Versão beta, baseado na web, com muitas características de comunidade virtual.

- **TellTable** em http://telltable-s.sourceforge.net/
  Precisa ser instalado em um servidor Unix ou Linux, usa o OpenOffice como processador de texto, mas abre uma janela do OpenOffice no navegador.

- **Web Collaborator** em http://webcollaborator.com/
  Gratuito se usado no servidor padrão, mas é pago para ser instalado em um servidor particular. Oferece um fórum para o grupo discutir o texto e histórico de versões além de RSS feed. Converte documento HTML para pdf.

- **Writely** em http://www.writely.com/
  Versão beta, multi-plataforma, requer as últimas versões dos navegadores. Editor “semi-síncrono”, porque fica constantemente atualizando a tela, de dois em dois minutos, ainda aceita *uploads* de documentos do MS Word para edição e salva como o formato doc. Usa o navegador como processador de texto.

3.2 Comparativo das ferramentas

A Tab. 1 faz um comparativo entre as capacidades dentre algumas ferramentas pesquisadas e o CoWS.

Tab. 1. estratégias de escrita colaborativa dos sistemas

Sistemas	Escrita	Edição	Revisão	Estratégia
EquiText	Online	Online	Online	Baseado na web incluindo escrita no navegador
SparrowWeb	Online	Online	Offline	Baseado na web excluindo escrita no navegador
Microsoft Office 2000, DocReview	Offline	Offline	Online	Baseado na web revisão apenas
CoWS	Online/Offline	Online/Offline	Online/Offline	Multi-síncrono, baseado em espaço de tuplas

4 CoWS e Ontologia de Contexto

Neste capitulo é explicado o funcionamento do CoWS, tanto a teoria por trás da criação do CoWS como também as suas funções. Além disso, é apresentada a ontologia de contexto utilizada para inserir contexto na ferramenta.

4.1 CoWS

O CoWS (Vieira, 2005) foi criado com o intuito de ser uma ferramenta de escrita colaborativa para sistemas distribuídos e móveis. Uma dificuldade inerente em criar sistemas distribuídos é o forte acoplamento desses sistemas. Para criar sistemas distribuídos mais facilmente foi proposto um *middleware* chamado Linda (GELERNTER, 1985), baseado no modelo de espaço de tuplas. Com o uso do Linda, o programador se concentra nos detalhes da aplicação em si e as questões de comunicação são tratados pelo *middleware*. Além disso, Linda possui um baixo acoplamento (desacoplamento espacial e temporal). Como o foco do Linda era sistemas distribuídos e não móveis, posteriormente foi criado o Lime (*Linda in Mobile Environments*) que é uma expansão do Linda especialmente feita para facilitar o desenvolvimento de sistemas distribuídos. O CoWS foi criado usando o Lime.

4.1.1 Linda

O modelo computacional do Linda foi proposto no início da década de oitenta como um novo modelo de comunicação entre processos (MURPHY et al., 1999) apud (VIEIRA et al., 2005c). Nesse modelo, os processos se comunicam através da escrita, leitura e remoção de dados em uma área de memória compartilhada chamada espaço de tuplas. Um espaço de tuplas é um repositório global e persistente de tuplas que são estruturas de dados constituídas por uma seqüência ordenada de campos. As tuplas contêm as informações reais que estão sendo comunicadas. Cada processo concorrente no sistema tem acesso ao espaço de tuplas (propriedade global), que existe independentemente da existência dos processos (propriedade de persistência). O modelo do Linda fornece coordenação entre os componentes por meio do compartilhamento do seu espaço de tuplas. Nesse modelo há o desacoplamento espacial e temporal. O desacoplamento temporal estabelece que os componentes não precisam coexistir ao mesmo tempo no sistema para se comunicar, o desacoplamento espacial

especifica que um dado componente pode residir em qualquer lugar do sistema distribuído.

Como as tuplas são anônimas, a sua pesquisa se dá através da observação de seus conteúdos por meio de um *template*. Um *template* é uma tupla cujos campos contêm valores reais ou formais. Valores reais especificam qualquer valor que pode ser armazenado em um dado campo ao passo que os valores formais atuam como curingas representando, assim, uma faixa de valores possíveis para um dado campo de acordo com o seu tipo. Na seleção de uma tupla os valores do *template* são comparados aos valores das tuplas armazenadas até que uma tupla com valores idênticos seja encontrada.

A manipulação das tuplas no espaço é feita com apenas três operações: escrever uma tupla no espaço (*out*), ler uma tupla do espaço de acordo com um *template* (rd) e retirar uma tupla do espaço de acordo com um *template* (*in*). No caso de existirem várias tuplas correspondentes a um dado *template*, apenas uma é retornada de forma não determinística. As últimas duas operações (ler e retirar) são bloqueantes, isto é, se uma tupla não for encontrada, o processo que solicitou é suspenso até que uma tupla que satisfaça a pesquisa apareça no espaço. No caso de múltiplos processos bloqueados para retirar uma tupla, quando a tupla aparece, apenas um processo selecionado de forma aleatória receberá a tupla. Os demais receberão uma cópia da tupla.

O Lime é uma evolução do Linda proposto por Murphy *et al* (MURPHY e PICCO, 2004), com suporte a ambientes móveis.

4.1.2 Lime

O Lime (*Linda in a Mobile Environment*) é um *middleware* projetado para possibilitar o desenvolvimento rápido de aplicações que apresentam mobilidade física (hosts) e/ou lógica (agentes) (MURPHY et al., 2000) apud (VIEIRA et al., 2005c).

Basicamente a diferença entre o Lime e o Linda é que no modelo de coordenação e comunicação do Lime é acrescentado tanto aa mobilidade mobilidade física (*hosts*) quanto a lógica (agentes).

Neste modelo, agentes são os únicos componentes ativos do sistema e os únicos que possuem um espaço de tuplas real. Já os *hosts* são apenas repositórios para os agentes ficando responsáveis por dar suporte à conexão e execução. Os agentes podem se mover através de hosts móveis que podem se

mover através do espaço físico. As propriedades do ambiente de um componente móvel forma o seu contexto. Quando a mobilidade é totalmente explorada não há contexto fixo, global e predefinido para a computação como assumido por Linda (MURPHY et al., 2000) apud (VIEIRA et al., 2005c). Em vez disso há um contexto global e dinâmico definido pelos diferentes contextos individuais das unidades móveis que estão conectadas. O modo como à unidade móvel interage nesse contexto dinâmico não muda porque operações básicas ocorrem usando as mesmas primitivas do Linda.

Espaço de Tuplas Federado

A presença de mobilidade impede a existência de um espaço de tuplas persistente e global, entretanto, cada agente móvel é dono de pelo menos um Espaço de Tuplas Lime (*Lime Tuple Space* – LTS). O compartilhamento do LTS é uma abstração que fornece à unidade móvel a ilusão de um espaço de tuplas local contendo tuplas de todas as unidades atualmente conectadas sem precisar saber quais são (MURPHY e PICCO, 2004) apud (VIEIRA et al., 2005c). Em outras palavras, uma operação *in* requisitada por qualquer um dos agentes pode retornar uma tupla correspondente do espaço de tuplas do Lime de qualquer agente conectado que possua aquela tupla em questão. Este espaço de tuplas tem o nome de espaço de tuplas federado que nada mais é do que a união de todas as tuplas de todos os hosts (Fig. 2).

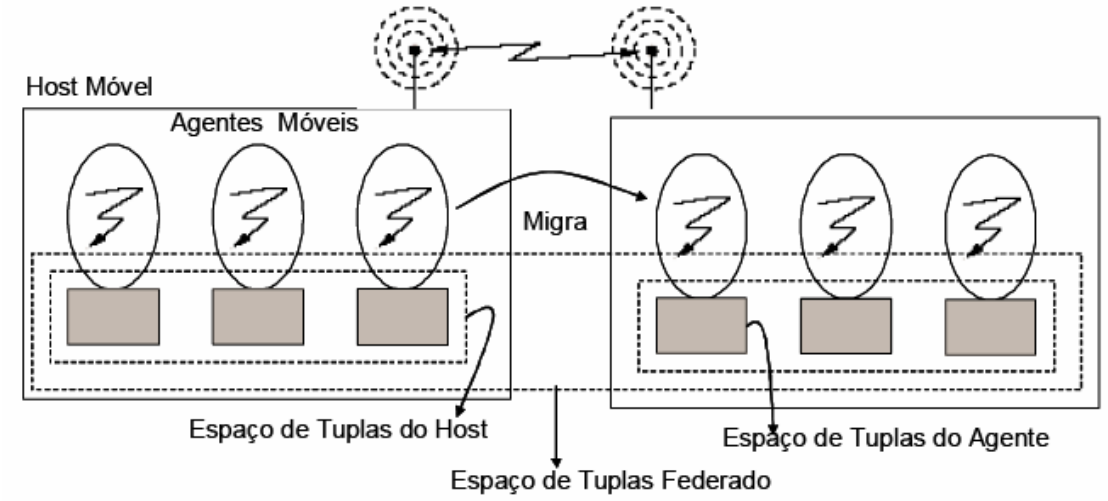

Fig. 2. Espaço de Tuplas Federado do Lime (VIEIRA et al., 2005b)

4.1.3 Editor Colaborativo e o Lime

Nesta subseção são apresentadas algumas questões que foram relevantes no projeto do CoWS (VIEIRA et al., 2005c) para adequá-lo ao Lime e discute como a abordagem de espaço de tuplas contribuiu para facilitar a implementação desse tipo de aplicação.

a) Papéis na Escrita Colaborativa

Um grupo formado para realizar escrita colaborativa pode ser composto por diversos papéis, como autor, revisor, coordenador, co-autor, editor, etc. A versão atual do CoWS apóia as atividades de múltiplos autores e co-autores.

b) Parágrafos e Tuplas

No editor, cada tupla armazenada no espaço de tuplas contém um parágrafo de um texto em edição. O conteúdo de um parágrafo pode ser uma seqüência de caracteres alfanuméricos e símbolos especiais do padrão ANSI. O documento compartilhado pelos autores no CoWS é um texto, composto por vários parágrafos que são as unidades atômicas de adição, edição e remoção. Essa coleção de parágrafos segue uma determinada ordem. Para armazenar cada parágrafo na ordem correta é preciso ter na tupla informações relacionadas à ordem dos parágrafos em relação ao texto. Essas e outras informações adicionais sobre as tuplas serão detalhadas no tópico f.

c) Modo Multi-síncrono e Modelo de Coordenação

No CoWS, os autores podem trabalhar engajados (modo síncrono) ou desengajados (modo assíncrono). No modo síncrono, as alterações dos autores serão percebidas tão logo ocorram. No modo assíncrono, as alterações realizadas por um autor só serão visíveis aos demais autores que estiverem conectados quando ele se engajar. Garantir o trabalho assíncrono é fundamental em cenários móveis onde a desconexão dos *hosts* é uma constante.

Os autores podem se conectar ao sistema, olhar o estado atual do texto, selecionar os parágrafos que desejam remover ou alterar e então se desconectar. Em modo assíncrono podem fazer suas alterações no texto, através da inclusão de novos parágrafos, remoção ou alteração dos existentes, que tenham sido previamente selecionados. Dessa maneira, o texto pode ser construído de forma independente e paralela e os resultados intermediários podem ser novamente

compartilhados quando a conectividade é restabelecida. Esta interação distribuída síncrona e assíncrona é obtida naturalmente por causa do modelo de coordenação dos espaços de tuplas: o total desacoplamento espacial e temporal.

d) Perspectiva Fracamente Consistente e Mobilidade

O editor exibe todos os parágrafos que estavam no espaço quando da sua última conexão mesmo os parágrafos que estiverem sendo editados por outros autores. Com isso, a perspectiva exibida no espaço de trabalho de um autor em modo assíncrono pode não ser totalmente consistente com o estado atual do texto, afinal ela representa a última informação conhecida sobre o texto.

e) Controle de Concorrência

As políticas de controle de acesso definem uma área exclusiva para interação do texto sem a interferência de outro usuário, que pode ser ao nível de seções, parágrafos ou letras, sendo que quanto menor o nível de granularidade definido, maior controle será necessário. O nível de granularidade do CoWS são os parágrafos e o tipo de controle de concorrência implementado é o bloqueio. O mecanismo de bloqueio, implementado pelo editor, funciona da maneira a seguir. O Lime possui suas operações estendidas com parâmetros de localização das tuplas, permitindo acesso a partes do espaço de tuplas compartilhado. Quando um parágrafo é criado, ele é inserido no espaço de tuplas local do agente que o criou. O agente que criou o parágrafo passa a ser o dono dele e detém o controle do mesmo. Quando é feita uma conexão ao espaço de tuplas federado, os espaços de tuplas dos demais agentes passam a enxergar esse parágrafo, porém sem poder modificá-lo. Caso um outro usuário deseje modificar ou remover o parágrafo bloqueado, ele deverá primeiro selecionar (tomar para si o controle) o parágrafo. Com isso, a tupla correspondente ao parágrafo é removida do espaço de tuplas onde está localizado fisicamente e é transferida para o espaço de tuplas local do agente que solicitou a seleção. Dessa maneira, o parágrafo passa a residir fisicamente nesse novo espaço de tuplas e operações realizadas por outros agentes não terão efeito sobre esse parágrafo. Porém, o bloqueio sobre ele é apenas parcial, porque todos os usuários têm a capacidade de selecionar qualquer parágrafo a qualquer momento, exceto os parágrafos que foram selecionados por um agente que está desconectado, sendo esse o único meio de bloquear totalmente o parágrafo.

f) Representação do Texto e Tuplas do Espaço

Uma das questões principais na utilização da abordagem de espaço de tuplas na modelagem do editor compartilhado CoWS é como montar o texto final a partir das diversas tuplas espalhadas pelo espaço, garantindo coerência com as intenções dos diversos autores. O espaço de tuplas não guarda nenhuma ordem das tuplas inseridas nem dos parágrafos, o que poderia levar a um texto desordenado na hora de remontagem do texto entre cada um dos usuários, pensando em diversos autores trabalhando de forma síncrona e assíncrona, modificando e produzindo diferentes versões do texto. Para resolver essa questão o CoWS propõe o uso de uma estrutura de dados que foi denominada lista não encadeada. Lista porque possui anterior e próximo. Não encadeada porque o anterior e próximo não são necessariamente os vizinhos imediatos. Com isso, o domínio do problema de inconsistência, que era distribuído, passa a ser apenas o de manter consistente uma lista local.

Logo, o modelo das tuplas que representam os parágrafos contém as seguintes informações:

- **Id**: Representa o identificador único de parágrafo possibilitando sua posterior recuperação do espaço de tuplas. Esse id é formado pelo *login* do usuário que o criou seguido de um seqüencial;
- **Head**: Indica o id do parágrafo de onde o parágrafo corrente foi criado. Null, se o parágrafo for inserido como o primeiro do texto;
- **Tail**: Armazena o id do parágrafo de referência anterior ou posterior ao parágrafo origem; Null, se o anterior ou posterior não existir;
- **Direction**: indica a direção de inserção do parágrafo dentro do texto. Assume os valores true/false. Se true, o Tail indica o parágrafo anterior. Se false, Tail indica o posterior;
- **Text**: O conteúdo do parágrafo;
- **Selection**: Indica o login do usuário que detém o controle do parágrafo;
- **Deleted**: Flag que indica se o parágrafo está ativo ou inativo no texto. Um parágrafo removido não é retirado imediatamente do espaço de tuplas. Ele fica marcado como inativo, permitindo que uma posterior recuperação seja realizada. Uma rotina de retirada de lixo pode de tempos em tempos, remover os parágrafos marcados como inativos.

g) Arquitetura do CoWS e Arquitetura do Lime

O CoWS possui uma arquitetura totalmente distribuída, com o texto global em produção sendo armazenado em diversos hosts e agentes, seguindo o padrão do Lime como mostra a Fig. 3. Cada agente detém seu espaço de tuplas local, onde suas tuplas são armazenadas. O conjunto dos espaços de tuplas locais engajados compõe o espaço de tuplas federado. Na parte de cima da figura é mostrado a arquitetura do CoWS que segue o modelo de arquitetura do Lime mostrada na parte de baixo.

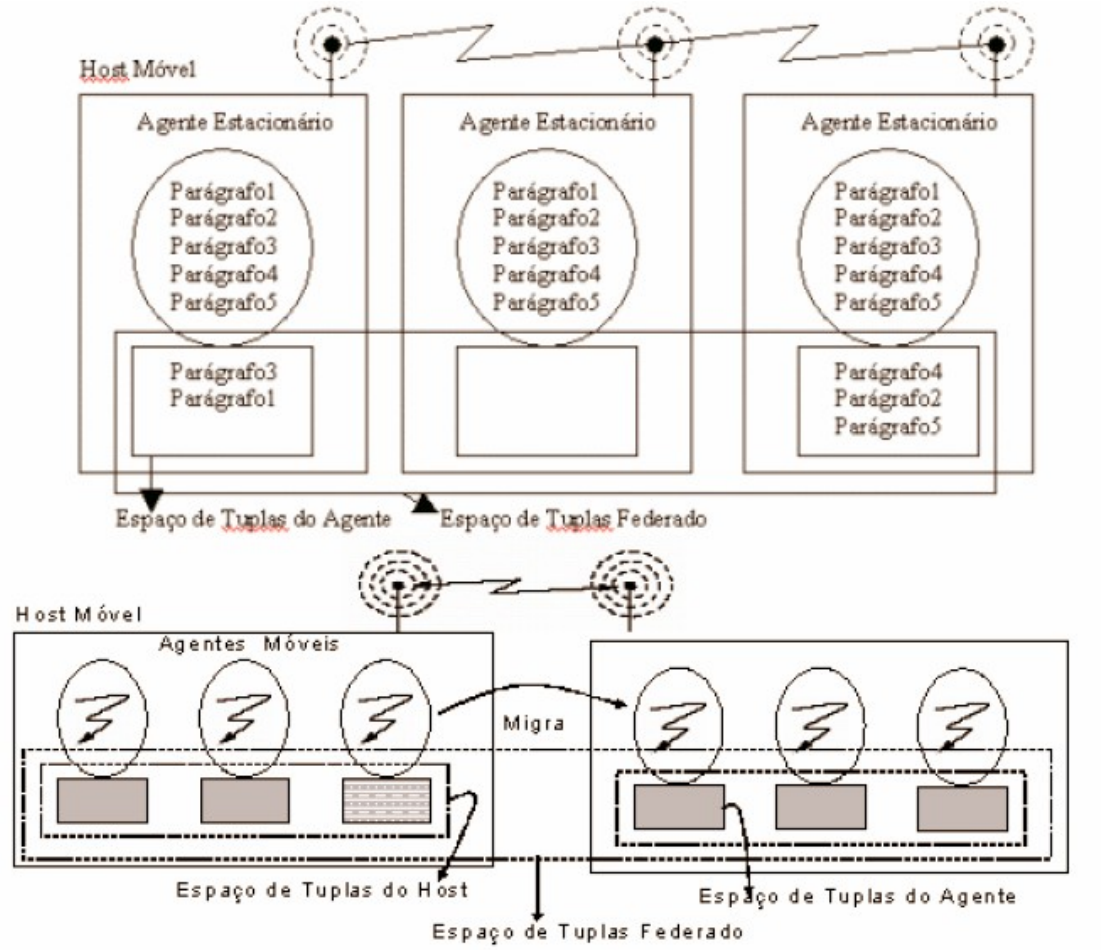

Fig. 3. Arquitetura geral de funcionamento do CoWS, figura de cima, seguindo o modelo do Lime, figura de baixo (VIEIRA et al., 2005b)

4.2 Ontologia de Contexto

A Ontologia de contexto usada neste trabalho é uma extensão da ontologia proposta em Vieira *et al*. (VIEIRA et al., 2005a) que será chamada OntoContext. OntoContext foi feita no Protégé na linguagem OWL DL.

4.2.1 Web Ontology Language (OWL)

OWL é uma linguagem para Web semântica desenvolvida pelo *World Wide Web Consortium* (W3C, 2006) com o objetivo de definir ontologias. Ontologias são descrições formais de conceitos de alguma área do conhecimento (classes), propriedades das classes que descrevem suas características, atributos das classes e restrições de propriedades (NOY e MCGUINNES, 2001). Ao contrário da linguagem HTML que é focada em formatação de texto e imagem e do XML que é simples demais para descrever bases de conhecimentos complexas, OWL possui características que a tornam ideal para a criação de ontologias, como: compatibilidade com padrões web para acessibilidade e internacionalização, fácil distribuição entre vários sistemas, escalabilidade e extensibilidade. Ontologias podem ser usadas não apenas em um aplicativo, mas sim em qualquer aplicativo da mesma área de conhecimento que precise e queira usar das facilidades que ontologias proporcionam. Por esses motivos OWL foi escolhida como a linguagem da ontologia de contexto utilizada no CoWS, a qual será explicada na próxima seção. OWL se subdivide em três linguagens (W3C, 2006), cada uma com um foco diferente .

- *OWL Lite* é a mais simples das linguagens, basicamente para quem precisa apenas de classificação e restrições simples, só suporta restrições de cardinalidade com valores 0 e 1.

- *OWL DL* (DL vem de lógica descritiva) suporta a maior expressividade possível levando em consideração a completude computacional (a todas as conclusões é garantido que são computáveis) e decidibilidade (todas as conclusões terminam em tempo finito). OWL DL possui todas as construções de OWL, porém com algumas restrições, por exemplo, uma linguagem pode ser uma subclasse de várias classes, mas uma classe não pode ser uma instância de outra classe. A OntoContext foi feita na linguagem OWL *DL*.

- *OWL Full* é usada por quem precisa do máximo de expressividade e a liberdade sintática de RDF, porém ao preço da falta de garantias computacionais.

4.2.2 OntoContext

A OntoContext é dividida em três conjuntos principais: *Context*, *Profile* e *Subject*.

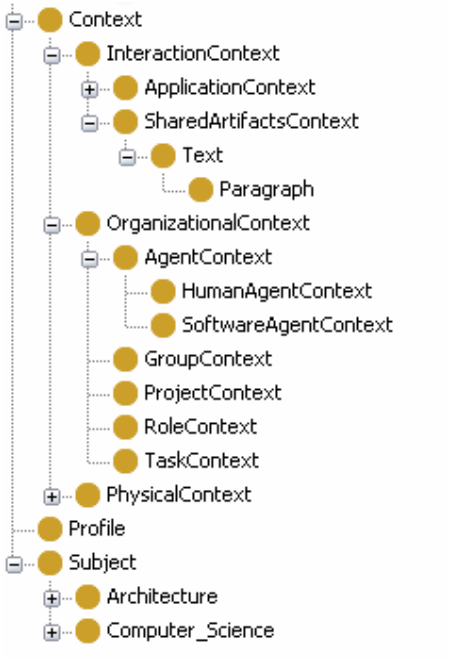

Fig. 4. A ontologia completa (Vieira et al.,2005a)

A seguir será explicado apenas as partes da OntoContext que foram usadas no CoWS. No artigo original que propõe a ontologia (VIEIRA et al., 2005a) existem mais detalhes sobre a ontologia completa mostrada na Fig. 4.

Context: A classe ***Context*** divide os diferentes tipos de informações de contexto que serão usadas. São três conjuntos disjuntos, ***InteractionContext***, ***OrganizationContext*** e ***PhysicalContext***. No caso do CoWS, apenas as informações dos conjuntos ***InteractionContext*** e ***OrganizationContext*** são usadas.

- ***InteractionContext*** contém informações relacionadas ao contexto de uma interação, o que vem acontecendo (síncrono) e o que aconteceu (assíncrono). Como subclasses de ***InteractionContext*** existem as classes ***ApplicationContext*** e ***SharedArtifactsContext***. ***SharedArtifactsContext*** é a subclasse que guarda os elementos relacionados a todo tipo de "coisa" que será compartilhada, por exemplo, dentro de ***SharedArtifactsContext*** temos uma subclasse ***Text*** que contém outra subclasse ***Paragraph***. No CoWS os textos e parágrafos serão as "coisas" que se tem interesse, por isso elas estão dentro de ***SharedArtifactsContext***. Essa extensão à classe ***SharedArtifactsContext*** serve para os requisitos do CoWS, porém podem variar de acordo com a aplicação.

  Cada instância de texto ou de parágrafo tem como propriedades específicas: ***textSubject*** e ***enfatize***. ***textSubject*** é constituído por uma instância da classe ***Subject***, a qual indica o assunto daquele texto/parágrafo. ***enfatize*** contém um conjunto de instâncias de ***HumanAgentContext***, pois ***enfatize*** representa quais usuários têm interesse sobre o texto/parágrafo ao qual a propriedade ***enfatize*** se refere.

- ***OrganizationContext*** contém informação sobre a organização onde a interação ocorre, por exemplo, o projeto, o grupo, as tarefas individuais ou em grupo, etc. Dentro de ***OrganizationContext*** tem a subclasse ***AgentContext***, que representa os elementos relacionados ao contexto dos membros do grupo. ***AgentContext*** se divide em ***HumanAgentContext*** e ***SoftwareAgenteContent***, que representam respectivamente membros humanos e membros computacionais do grupo (possivelmente um agente inteligente). As propriedades de ***HumanAgentContext*** que vão ser usadas são ***hasProfile***, que indica um ***Profile*** (perfil) da instância de usuário em questão e ***receiveNotify***, que armazena quais textos/parágrafos o usuário já foi notificado pelo sistema, evitando assim notificar o usuário mais de uma vez sobre o mesmo texto.

Profile: a classe **P*rofile*** representa um perfil do usuário do sistema, onde são guardadas suas configurações e preferências. ***Profile*** tem apenas dois atributos: ***hasInterest*** e ***notify***. O ***hasInterest*** contém um conjunto de objetos da classe ***Subject*** que indica os assuntos que o usuário tem interesse. ***notify*** é a

configuração do usuário que indica como o sistema deve se comportar com relação à notificação dos textos do sistema. Por exemplo, se ele quer ser notificado de todos os textos criados depois da sua última entrada no sistema, ou se ele quer ser notificado sobre textos que tenham sido cadastrados como tendo o mesmo assunto de sua preferência.

Subject: é uma sub-ontologia que contém objetos que representam um assunto qualquer, que pode ser categorizado por áreas do conhecimento: computação, medicina, etc.

As sub-ontologia **Subject** foi criada apenas para o projeto do CoWS, não fazia parte da proposta original, basicamente são usadas no CoWS as partes da ontologia relacionadas a **Text** e **HumanAgentContext** que são as partes usadas nas regras que serão explicadas no capitulo 5, junto com a nova arquitetura do CoWS.

5 Implementação

Este capítulo apresenta a arquitetura do projeto assim como detalhes de implementação do CoWS e das tecnologias usadas no projeto.

5.1 Arquitetura

A Fig. 5 mostra uma visão em alto nível da arquitetura dos componentes de contexto do CoWS, depois das alterações propostas neste trabalho. No CoWS propriamente dito, temos o projeto antigo intacto com a adição de perfis de usuário e a base de contexto. Quando essas informações dos perfis são registradas ou atualizadas, elas são captadas pelo agente Java que encapsula e as manda para a OntoContext. Essas informações são acrescentadas ao OntoContext e o Jena executa a máquina de inferência sobre a ontologia e as regras que foram criadas. Depois disso, são gravados as novas informações que foram inferidas na OntoContext, e o CoWS mostra ao usuário as informações inferidas que são passadas pelo agente Java.

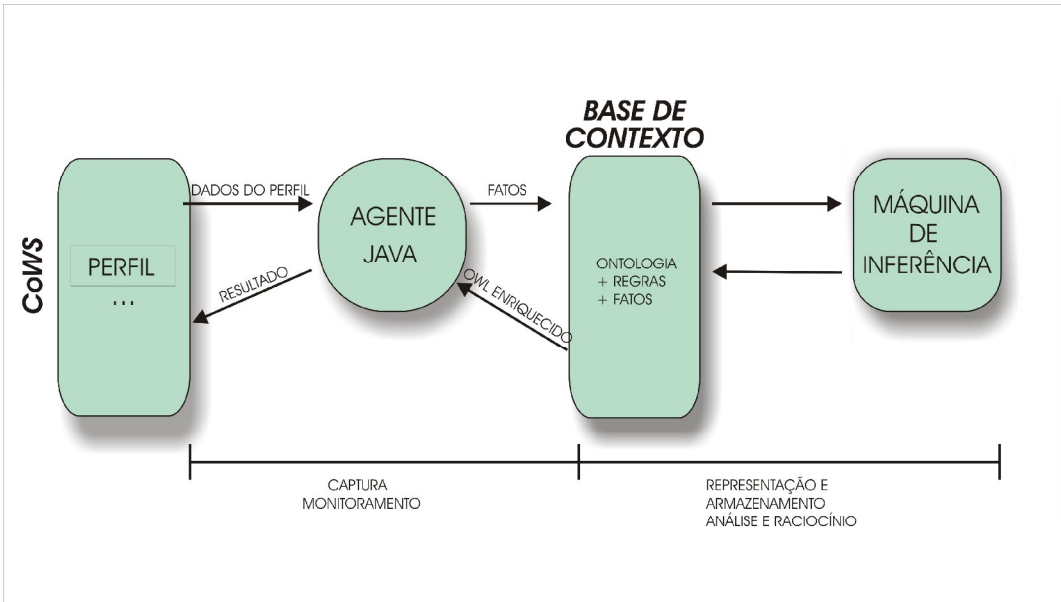

Fig. 5. Arquitetura dos Componentes de Contexto do CoWS

5.1.1 *Agente Java*

O processo de adquirir contexto é chamado *context acquisition* (DEY, 2000). Existem diversas maneiras de se adquirir informação contextual. Normalmente são usados sensores físicos (localização, temperatura...), mas no caso do CoWS foi usado um agente que funciona como um sensor lógico. Esse agente é iniciado junto com o CoWS e fica esperando que o usuário configure as informações do seu perfil. Quando o usuário insere suas preferências, o agente encapsula essas informações em uma classe Java que funciona como uma representação similar à ontologia. Depois de criados os objetos eles são passados para o Jena, que será explicado na próxima seção.

5.1.2 *Jena*

Jena é um *framework* Java, *open source,* desenvolvido pelo HP Labs Semantic Web Programme. Com o objetivo de construir aplicações com suporte à Web Semântica. O Jena suporta três tipos de linguagens ontológicas: RDF, DAML+OIL e OWL, incluindo uma máquina de inferência baseada em regras. Algumas das capacidades do Jena:

- API para RDF
- Lê e escreve RDF em vários formatos dentre eles RDF/XML, e N-Triples (formato mais fácil para pessoas lerem)
- RDQL **–** linguagem de consulta para RDF
- API para OWL
- Armazenamento em memória ou de forma persistente

Neste projeto, o Jena foi usado para ler e tratar a OntoContext. Depois que o agente envia os objetos que representam o perfil do usuário, essas informações são inseridas no OntoContext sobre a qual é executada a máquina de inferência, detalhada na próxima seção.

Máquina de inferência

O sistema de inferência do Jena permite que diferentes raciocinadores sejam conectados ao Jena. Esses raciocinadores são usados para derivar assertivas adicionais de uma base RDF (OWL é baseada em RDF, por isso é considerada uma base RDF) a partir de qualquer informação ontológica opcional mais axiomas e regras associadas ao raciocinador. O uso primário deste mecanismo é para permitir que sejam inferidos fatos adicionais das instâncias e descrições de classe em linguagens tais como RDFS e OWL.

A Fig. 6 mostra uma visão geral do suporte à máquina de inferência do Jena. O agente acessa a máquina de inferência usando o ModelFactory. Esse Modelfactory associa uma base de dados (Ontologia) com um raciocinador (*reasoner*), criando um novo modelo. Quaisquer consultas que sejam feitas ao modelo, após ele ter sido integrado a um raciocinador, trarão como resultados todos os dados originais da ontologia e também todas as assertivas derivadas pelo raciocinador através das regras definidas e das propriedades definidas na ontologia.

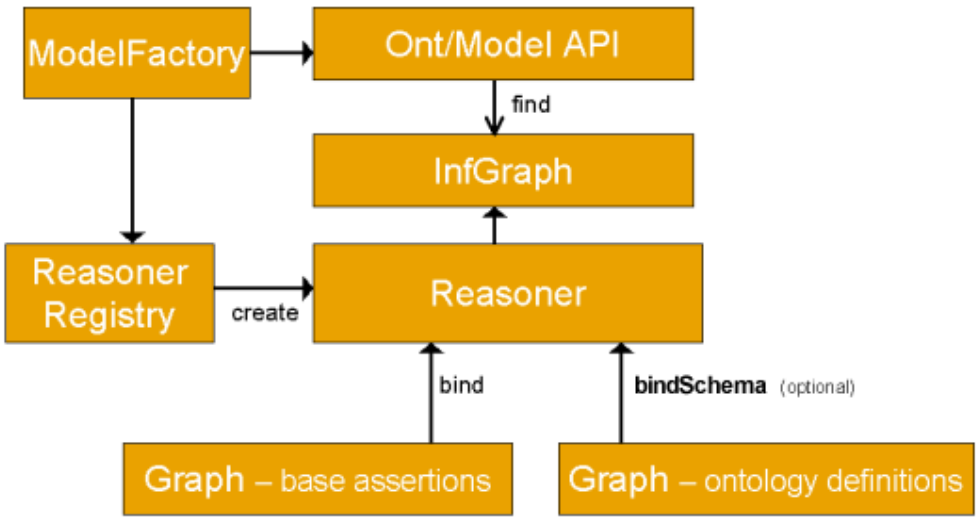

Fig. 6. Arquitetura geral da máquina de inferência do Jena (JENA - A SEMANTIC WEB FRAMEWORK FOR JAVA, 2006)

O Jena provê várias interfaces da classe Model que podem ser usadas para ligar as regras aos dados. Dentre essas interfaces no projeto do CoWS foi usado a classe InfModel que implementa da interface Model. Essa classe permite que se façam várias operações de controle do resultado da união do raciocinador com os dados.

Cada *raciocinador* pode ser ligado a diferentes esquemas (Schemas) e instâncias, por meio dos métodos *bind* e *bindSchema*. OWL não separa fortemente dados de Schema, por isso essa necessidade de se ligar um *raciocinador* a um conjunto de instâncias apenas (ignorando o Schema) usando o bindSchema e o inverso também (ligar o *raciocinador* com apenas o Schema de uma ontologia sem dados) usando o bind.

O *Reasoner*Registry é uma classe estática que disponibiliza todos os raciocinadores padrões do Jena e com ela podem ser criados e registrados novos raciocinadores. Na Fig. 7 está um código exemplificando como se faz para criar um objeto InfModel que é a união de um Model (dados) com um GRR (raciocinador), com esse InfModel é que são realizadas todas as operações.

```
//conjunto de dados que vem do OntoContext
 Model dados = FileManager.get().loadModel("OntoContext.owl");
 List regras = Rule.rulesFromURL("regrasCoWS.rules");
 GenericRuleReasoner reasoner = new GenericRuleReasoner(regras);
//InfModel estende o ModelFactory que possibilita mais controle e
acesso ao //grafo resultante da união dos dados com as regras
InfModel inf = ModelFactory.createInfModel(reasoner, dados);
```

Fig. 7. Código Fonte para Inicialização do Agente Java

Na Tab. 2 está uma pequena explicação sobre todos os raciocinadores padrão existentes no Jena. O Generic Rule *Reasoner* (GRR) foi o raciocinador escolhido no projeto do CoWS.

Tab. 2. Raciocinadores do Jena (JENA - A SEMANTIC WEB FRAMEWORK FOR JAVA, 2006)

Transitive *reasoner*	Fornece suporte para armazenar e percorrer classes e propriedades ligadas. Implementa apenas as propriedades transitive e symmetric (transitiva e simétrica) de rdfs:subPropertyOf e de rdfs:subClassOf
RDFS rule *reasoner*	Implementa um subconjunto configurável das funções de RDFS
OWL, OWL Mini, OWL Micro *Reasoners*	Um conjunto de implementações, ainda incompleto, da linguagem OWL/Lite
DAML micro *reasoner*	Usado internamente para habilitar a API legada DAML. Provê uma capacidade mínima de inferência para DAML
Generic rule *reasoner*	Raciocinador baseado em regras que suporta a criação de regras definidas pelo programador. Suporta encadeamento para frente (*forward*

	chaining), encadeamento para trás (*tabled backward chaining)* e estratégias de execução híbridas

5.1.3 Generic Rule *Reasoner*

O GRR é o mais independente dos raciocinadores. Ele foi usado para implementar tanto o *raciocinador* RDFS quanto o OWL. Diferentemente dos outros raciocinadores, o GRR baseia-se em regras que seguem uma sintaxe própria. Além disso, é possível importar todas as funções dos outros raciocinadores, o que torna o GRR o mais poderoso dos raciocinadores. Toda a inferência gerada pelo CoWS foi baseada no GRR, mas também com algumas funções importadas do raciocinador OWL.

Uma regra do GRR é definida como sendo uma instância da classe Rule que contém uma lista de premissas e conclusões sobre essas premissas, opcionalmente uma regra possui um nome e um sentido (frente ou trás). Uma premissa ou uma conclusão pode ser apenas uma tripla, uma tripla estendida ou uma chamada a procedimento primitivo (chamado *builtin*). Um conjunto de regras é a lista de todas as regras. O Jena tem um *parser* para checar a legalidade das regras definidas seguindo a sintaxe original. Esse *parser*, porém, pode ser substituído por um *parser* definido pelo usuário, já que ele é muito fraco no diagnóstico de erro. Durante a implementação deste trabalho foi possível perceber que esse *parser* só percebe os erros de sintaxe e com descrições erradas dos erros. Isso dificultou a criação das regras da máquina de inferência, e é um ponto primordial a um maior suporte do Jena em relação à definição de regras.
A sintaxe das regras aceitas pelo GRR aparece na Fig. 8.

```
Rule     :=   bare-rule .
          or   [ bare-rule ]
               or   [ ruleName : bare-rule ]
bare-rule :=   term, ... term -> hterm, ... hterm  // frente
          or   term, ... term <- term, ... term    // tras
hterm     :=   term
          or   [ bare-rule ]
term      :=   (node, node, node)           // uma tripla
          or   (node, node, functor)        // tripla estendida
          or   builtin(node, ... node)      // procedimento primitivo
functor   :=   functorName(node, ... node)  // literal estruturado
node      :=   uri-ref                      // http://foo.com/eg
          or   prefix:localname            // rdf:type
          or   ?varname                    // variável
          or   'a literal'                 // string
          or   'lex'^^typeURI              // literal tipificado
          or   number                      // 42 ou 25.5
```

Fig. 8. Sintaxe das regras aceitas pelo GRR (JENA - A SEMANTIC WEB FRAMEWORK FOR JAVA, 2006)

Uma das regras criadas no projeto é mostrada na Fig. 9. Essa regra checa se o usuário do sistema possui uma configuração no CoWS que indique o desejo de ser avisado de novos textos criados no sistema que tenham o mesmo assunto que o usuário indicou como sendo assuntos de seu interesse. Se um texto for cadastrado com o assunto do interesse do usuário essa regra vai criar uma nova instância de **enfatize** dentro da classe **Text** com esse usuário. O *makeInstance* é um procedimento *builtin* do GRR.

```
//declaração do namespace da OntoContext
@prefix pre: <http://www.owl-ontologies.com/OntoContext.owl#>.
//importando as funções do OWL reasoner
@include <OWL>.
[rule1:
(?perfil pre:hasInterest ?interesse)
(?perfil rdf:type pre:Profile)
(?perfil pre:notify 'NOTIFYAREAOFINTEREST')
(?usuario rdf:type pre:HumanAgentContext)
(?usuario pre:hasProfile ?perfil)
(?texto rdf:type pre:Text)
(?texto pre:textSubject ?assuntotexto )
equal (?assuntotexto, ?interesse )
>
makeInstance (?texto, pre:enfatize, ?usuario )]
```

Fig. 9. Regra criada para identificar textos cujo assunto seja de interesse do usuário

Além do *makeInstance* existem vários outros *builtin*, os quais são detalhados na Tab. 3.

Tab. 3. Procedimentos primitivos (builtin) do GRR (JENA - A SEMANTIC WEB FRAMEWORK FOR JAVA, 2006)

Builtin	Operações
`isLiteral(?x)` `notLiteral(?x)` `isFunctor(?x)` `notFunctor(?x)` `isBNode(?x) notBNode(?x)`	Testa se o argumento é ou não um literal, um functor ou um nó vazio.
`Bound(?x...)` `unbound(?x..)`	Testa se todos os argumentos foram ou não inicializados.
`equal(?x,?y)` `notEqual(?x,?y)`	Testa se x=y (ou x != y). É um teste semântico, ou seja ele considera que xsd:int 1 e xsd:decimal 1 são iguais.
`lessThan(?x, ?y),` `greaterThan(?x, ?y)` `le(?x, ?y), ge(?x, ?y)`	Testa se x é <, >, <= or >= y. Só funciona para os tipos númericos e temporais.
`sum(?a, ?b, ?c)` `addOne(?a, ?c)` `difference(?a, ?b, ?c)` `product(?a, ?b, ?c)`	Atribui à variável ?c o valor (a+b), (a+1) (a-b) ou (a*b).
`makeTemp(?x)`	Liga ?x a um novo nó em branco.
`makeInstance(?x, ?p, ?v)` `makeInstance(?x, ?p, ?t, ?v)`	Aponta ?v para um nó em branco o qual é atribuído com o valor da propriedade ?p no recurso ?x com opcionalmente o tipo ?t.
`noValue(?x, ?p)` `noValue(?x ?p ?v)`	Retorna True se não existe tripla (x, p, *) conhecida ou se (x, p, v) não existe no modelo.
`remove(n, ...)` `drop(n, ...)`	Remove a tripla que causou o enésimo termo que ativou essa regra. A remoção vai se propagar pelas outras regras posteriores. O *drop* faz a mesma coisa exceto que a remoção não ativa nenhuma regra posterior.
`isDType(?l, ?t)` `notDType(?l, ?t)`	Testa se o literal ?l é ou não uma instância do tipo ?t.
`print(?x, ...)`	Imprime os argumentos.
`Table(?p) tableAll()`	Declara todos as relações envolvendo a propriedade ?p. tableAll declara todos as relações.

5.1.4 Mecanismos de ativação de regras

O GRR possui suporte a três tipos de mecanismos de ativação de regras, o encadeamento para frente, encadeamento para trás e um modo híbrido (Tab. 2).(JENA - A SEMANTIC WEB FRAMEWORK FOR JAVA, 2006)

Encadeamento para frente

Se o raciocinador é configurado para rodar em modo para frente apenas, na primeira vez que o modelo de inferência recebe uma consulta, todos os dados relevantes do modelo são enviados para o mecanismo de regras. Caso uma regra cause a criação de triplas extras, podem ser ativadas novas regras. Se as regras não forem bem pensadas isso pode acarretar em *loop* infinito. Cada vez que são criadas ou removidas triplas do modelo pelos próprios métodos da API, as regras podem ser ativadas. A inferência acaba quando nenhuma regra for mais ativada. Quando estiver funcionando no modo para frente todas as regras são consideradas como sendo para frente mesmo que elas estejam definidas com a sintaxe para trás ("<-"). O algoritmo usado no mecanismo encadeamento para frente é o RETE (FORGY, 1982) que trabalha incrementalmente e aplica todas as regras possíveis ele é rápido mais em contrapartida ocupa muito espaço na memória.

Encadeamento para trás

No modo para trás o raciocinador usa um mecanismo de *logic programming* (LP) como uma estratégia de execução parecida com o mecanismo de Prolog. Quando o modelo de inferência recebe uma consulta, essa consulta é transformada em um objetivo, que a máquina de inferência tenta alcançar usando todas as triplas armazenadas e *backtracking*. O encadeamento para trás não trabalha incrementalmente e normalmente executa um menor número de operações para chegar ao mesmo resultado do modo para frente (pode ser pior em alguns casos).

Híbrido

O raciocinador de regras pode ser configurado para usar ambos os mecanismos (frente e trás) em conjunto, quando o modo híbrido é escolhido o fluxo percorrido pela regra acontece como na Fig. 10.

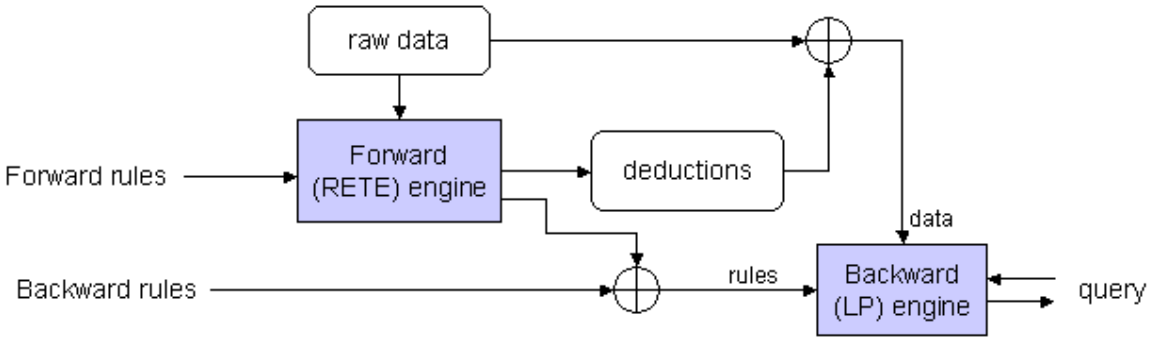

Fig. 10. fluxo dos dados no modo híbrido (JENA - A SEMANTIC WEB FRAMEWORK FOR JAVA, 2006)

O mecanismo para frente executa primeiro e guarda um conjunto de premissas na área de deduções. Todas as regras para frente que porventura criem novas regras para trás vão instanciar-las de acordo apenas com as variáveis guardadas nas deduções e depois vão passar as regras instanciadas para o mecanismo LP para trás.

Todas as consultas são resolvidas posteriormente pelo *LP engine* usando a mistura dos dados brutos e das deduções que vieram do mecanismo para frente. O modo híbrido possui vantagens sobre o modo para frente porque gasta menos memória e não tem o problema da lentidão (no pior caso) como o modo para trás, porém criar regras que realmente funcionem direito no modo híbrido é um pouco mais complicado do que nos outros modos.

5.2 Exemplo de uso

Quando o usuário entra no sistema ele preenche seu login e senha. Dentro do sistema a qualquer momento o usuário pode configurar suas preferências (Fig. 11) no menu Exibir > Preferências. Nas preferências o usuário pode configurar o sistema, para que quando ele se logar apareça uma notificação dos textos que sejam do seu interesse, por exemplo, ele pode querer que todos os textos que tenham como assunto “inteligência artificial” sejam mostrados a ele. Ao se logar, ele pode configurar o sistema também para mostrar os textos criados ou modificados após a sua última entrada no sistema, ou os textos de um usuário especifico.

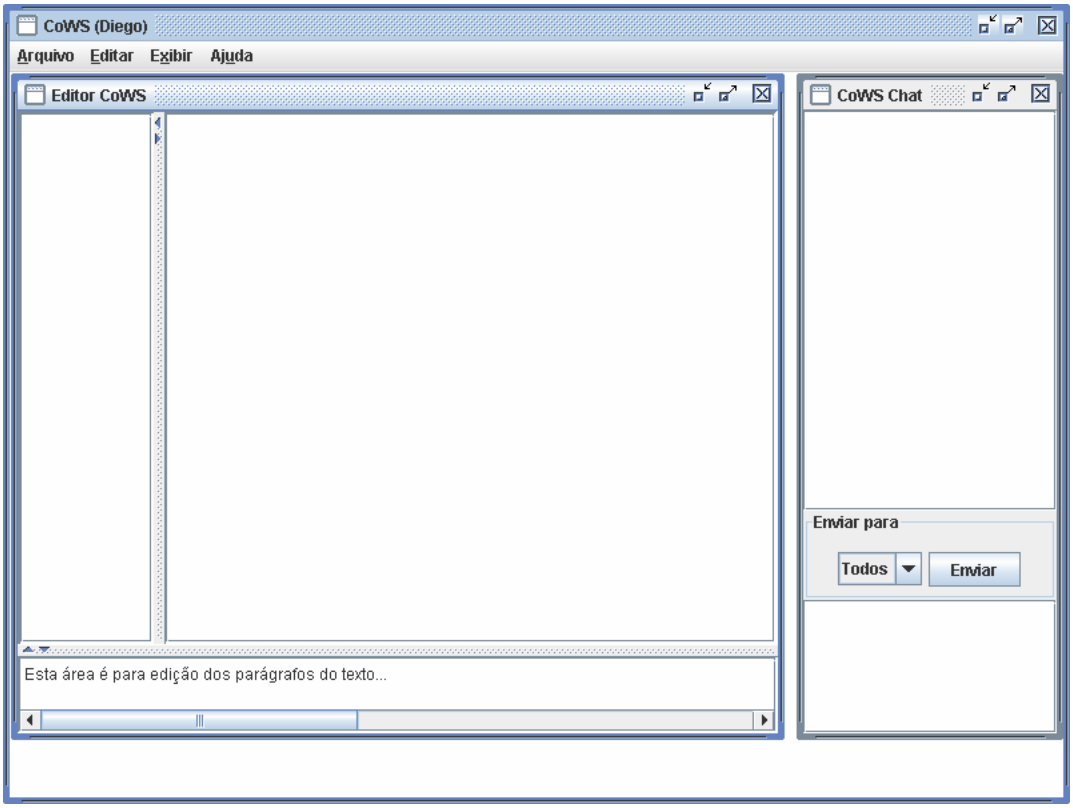

Fig. 11. tela inicial do CoWS

Ao mesmo tempo em que os textos que o usuário trabalha, seu perfil e suas preferências vão sendo cadastrados no sistema e também na ontologia. A Fig. 12 mostra uma instância do usuário Carolina na OntoContext.

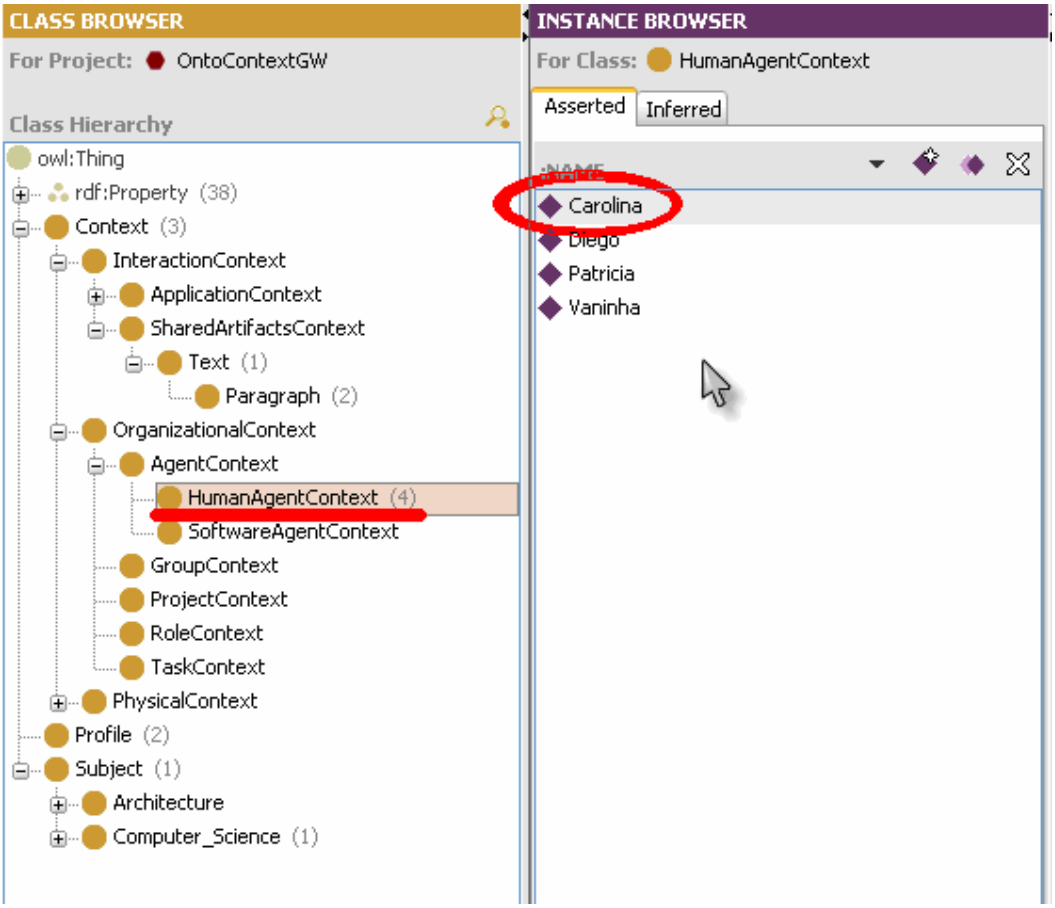

Fig. 12. No Protégé, uma instância de Carolina de HumanAgentContext

Carolina cadastrou no CoWS um texto com o assunto Databases, após o usuário Diego configurar seu perfil, o agente Jena recebe suas preferências, coloca-as no OntoContext e executa a máquina de inferência. Esta percebe que o usuário Diego está interessado nos assuntos Databases (Fig. 13).

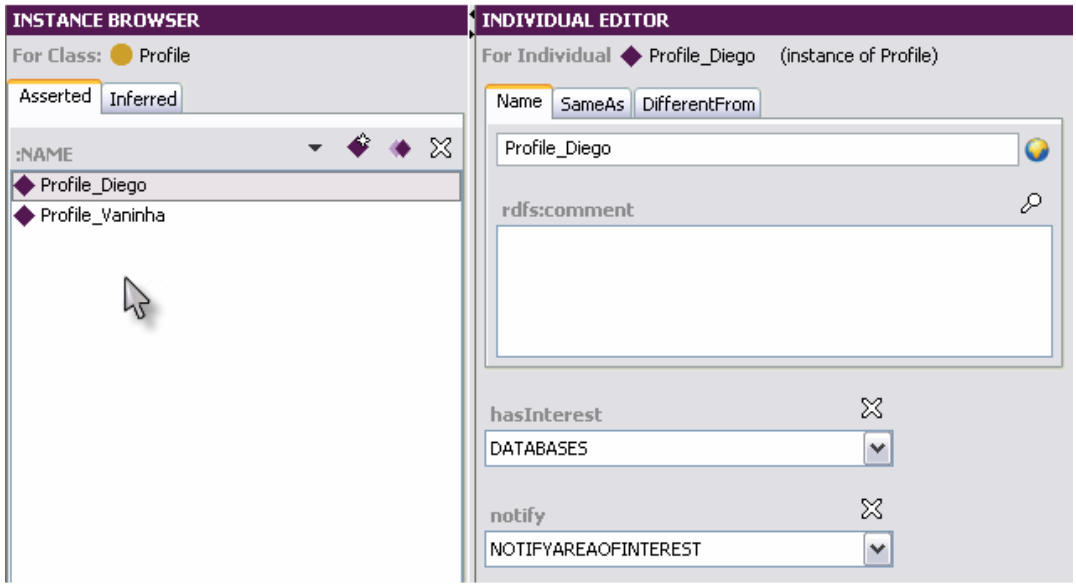

Fig. 13. Perfil do usuário Diego

A máquina de inferência percebe também que o usuário Carolina inseriu um texto com o assunto Databases que ainda não foi mostrado ao usuário Diego e insere na ontologia, dentro do Text correspondente ao texto que Carolina cadastrou o enfatize com o usuário Diego (Fig. 14).

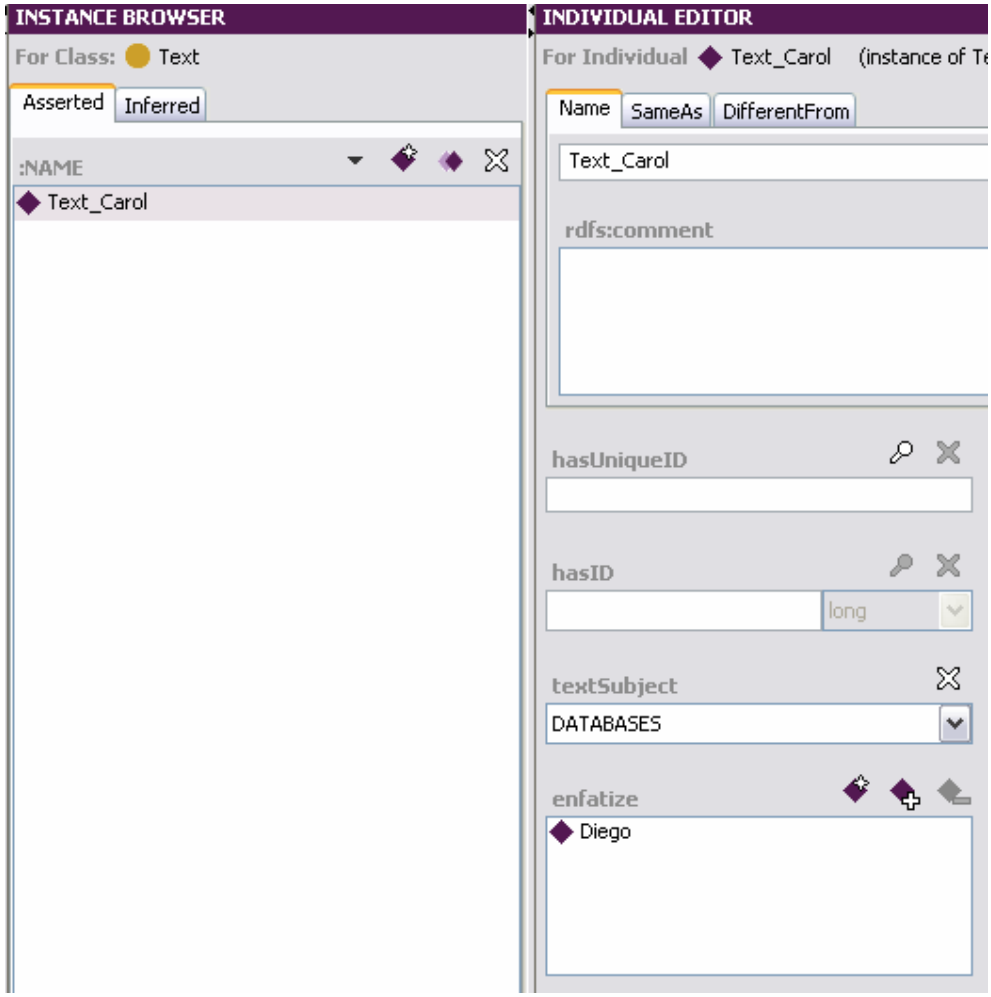

Fig. 14. Após a execução das regras de inferência, o usuário Diego é inserido no OntoContext

5.3 Resultados e dificuldades encontradas

Depois de todo esse processo, o agente recebe do Jena as informações que vão ser de interesse do usuário e envia para o CoWS o resultado de toda essa inferência realizada no OntoContext. O CoWS recebe e mostra uma nova janela ao usuário com os textos que ele pode ter interesse desde que ele ainda não tenha sido avisado desses textos (o usuário não recebe um aviso sobre o mesmo texto duas vezes).

As maiores dificuldades na implementação foram relativas ao funcionamento do Jena, devido à documentação fraca e aos vários *bugs* encontrados (alguns sem solução ainda). O tempo de criar o agente e a integração

com o CoWS foi muito curto e o resultado significativo como aplicação prática de toda essa teoria abordada.

6 Conclusão e Trabalhos Futuros

Foi mostrado nesse projeto que o modelo de abordagem orientado a ontologias é viável para a construção de implementações cientes de contexto. Não porque seja mais fácil criar aplicações usando um *framework* baseado em ontologias e regras de inferência do que em relação a uma programação pura, porque com certeza não é mais fácil. Em projetos pequenos, com regras simples, bem definidas e imutáveis seria muito mais rápido e fácil para um programador experiente criar diretamente em uma linguagem de programação como Java uma aplicação do mesmo nível do CoWS. Porém, não é assim que funciona na vida real. Contexto é um conceito muito subjetivo e as regras que regem uma aplicação ciente de contexto podem ser complexas e podem estar em constante evolução. Nesse caso, uma aplicação em C++ ou Java puro mostraria toda a sua fraqueza, pois seria necessário um esforço muito maior para a manutenção do sistema. Esse problema é menor em uma aplicação baseada em ontologias que possui uma grande capacidade de expansão e escalabilidade devido às características inerentes da OWL.

Como as ferramentas existentes para lidar com ontologias e o próprio Jena ainda são muito recentes, elas ainda têm muitos problemas. Mas com o amadurecimento dessas ferramentas, abordagens orientadas a ontologias devem se popularizar devido a suas características ideais para vários tipos de aplicações em um mundo totalmente conectado à web. O melhor de tudo é que pela própria característica do OWL, sistemas como o CoWS podem ficar obsoletos, mas mesmo assim a sua contribuição não se perde porque sua ontologia pode ser absorvida por sistemas similares que venham a substituí-lo no futuro.

As regras ontológicas usadas no CoWS são bastante simples. Em algum trabalho futuro o CoWS pode ser melhorado com mais regras e possíveis configurações do perfil do usuário, além de melhorias na interface e na capacidade do editor. O raciocinador do Jena também é um ponto passível de ajustes, pois ainda deixa muito a desejar em desempenho. Para executar apenas uma regra simples em uma ontologia com cerca de uma dezena de instâncias, o raciocinador chegava a levar 5 minutos em um computador com 2Ghz e 512MB de RAM. Na própria documentação do Jena era recomendada a utilização de raciocinadores

externos. Qualquer *raciocinador* que siga o padrão DIG é suportado pelo Jena; Pellet (PELLET, 2006), Racer{racer, 2006 276 /id} são os mais famosos.

Referências Bibliográficas

ALARCON , R. A., GUERRERO, L. A., OCHOA, S. F. P. J. A., 2005, "Context in Collaborative Mobile Scenarios", *Workshops on Cooperative Systems and Context, and, Groupware and Context*, v. 133.

ARAÚJO, R. M., 2000, *Ampliando a Cultura de Processos de Software - Um Enfoque Baseado em Groupware e Workflow*, Tese de D.Sc., COPPE/UFRJ, Rio de Janeiro.

AXT, M., COSTA, A. C. R., MARTIS, A. R., 2005, ***Suporte À Cooperação Em Ambiente Virtual De Escrita Coletiva***. Universidade Federal do Rio Grande do Sul (UFRGS).

BRÉZILLON, P., BORGES, M. R. S., PINO, J. A., POMEROL, J.-C., 2004, "Context-Awareness in Group Work: Three Case Studies". In: *IFIP International Conference on Decision Support Systems (DSS-2004). Decision Support in Uncertain and Complex World.*, Italia.

CHEN, G., KOTZ, D., 2000, *A Survey of Context-Aware Mobile Computing Research*. Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College.

CHEN, H., FININ, T., JOSHI, A., 2003, "An Ontology for Context-Aware Pervasive Computing Environments". In: ***Workshop on Ontologies and Distributed Systems, Special Issue in 8th IJCAI***, Acapulco, Mexico.

CHEN, H., FININ, T., JOSHI, A., PENG, Y., 2005, "UMBC eBiquity Project: Context Broker Architecture (CoBrA)", Access in 12/2005.

DEY, A. K., 2000, *Providing Architectural Support for Building Context-Aware Applications*, PhD, Georgia Institute of Technology.

DEY, A. K., ABOWD, G. D., 2000, "Towards a Better Understanding of Context and Context-Awareness". In: *Proceedings of the CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness*, The Hague, Netherlands.

DEY, A. K., SALBER, D., ABOWD, G. D., 2001, "A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications", *Human Computer Interaction Journal*, v. 16, n. Special Issue on Context-Aware Computing, pp. 97-166.

DILLENBOURG, P., BAKER, M. J., BLAYE, A., O'MALLEY, C., 1996, "The evolution of research on collaborative learning", *Learning in Humans and Machines: Towards an Interdisciplinary Learning Science*, Oxford: Pergamon.

DOURISH, P., BELLOTTI, V., 1992, "Awareness and Coordination in Shared Workspaces". In: *Proc. ACM Conference on Computer Supported Cooperative Work (CSCW'92)*, pp. 107-114, Toronto, Ontario.

ELLIS, C. A., WAINER, J., 1991, "Groupware and Computer Supported Cooperative Work".

ELLIS, L., GIBBS, S. J., REIN, G. L., 1991, "Groupware: Some Issues and Experiences", *Communications of the ACM*, v. 34, n. 1, pp. 38-58.

FORGY, C. L., 1982, "*RETE: A fast algorithm for the many pattern/many object pattern match problem*".

GELERNTER, D., 1985, "Generative Communication in Linda", v. ACM Computing Surveys. 7(1):80-112.

GONÇALVES, P. R., PADILHA, T. P. P., 2003, "Implementação de uma Ferramenta de Edição de Texto Colaborativa", v. Anais do V Encontro de Estudantes de Informática do
Tocantins, Palmas, TO., pp. 353-360.

GU, T., PUNG, H. K., ZHANG, D. Q., WANG, X. H., 2004a, "A Middleware for Building Context-Aware Mobile Services". In: *Proceedings of IEEE Vehicular Technology Conference (VTC 2004)*, Milan, Italy.

GU, T., WANG, X. H., PUNG, H. K., ZHANG, D. Q., 2004b, "An Ontology-based Context Model in Intelligent Environments". In: *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, San Diego, California, USA.

HARTER, A., HOPPER, A., STEGGELS, P., WARD, A., WEBSTER, P., 2002, "The Anatomy of a Context-Aware Application", *Wireless Networks*, v. 8, n. 2-3, pp. 187-197.

HELD, A., BUCHHOLZ, S., SCHILL, A., 2002, "Modeling of context information for pervasive computing applications". In: *Proceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics (SCI2002)*, Orlando, FL, USA.

HENRICKSEN, K., INDULSKA, J., RAKOTONIRAINY, A., 2003, "Generating Context Management Infrastructure from High-Level Context Models". In: *Industrial Track Proceedings of the 4th International Conference on Mobile Data Management (MDM2003)*, pp. 1-6, Melbourne/Australia.

HENRICKSEN, K., INDULSKA, J., RAKOTONIRAINY, A., 2002, "Modeling Context Information in Pervasive Computing Systems". In: *Proceedings of 1st International Conference on Pervasive Computing*, v. LNCS 2414, pp. 167-180, Springer, Zurich, Switzerland.

HORROCKS, I., 2002, "DAML+OIL: a Reason-able Web Ontology Language ". In: *Proceedings of the 8th
International Conference on Extending Database Technology (EDBT)*, Prague.

JENA - A SEMANTIC WEB FRAMEWORK FOR JAVA, 2006. In: http://jena.sourceforge.net/.

KINDBERG, T., BARTON, J., 2001, "A Web-based Nomadic Computing System", *Computer Networks*, v. 35, n. 4, pp. 443-456.

MERRIAN WEBSTER ONLINE, 2005. In: http://www.webster.com/.

MURPHY, A. L., PICCO, G. P., ROMAN, G.-C., 2000, "Developing Mobile Computing Applications with Lime", v. 22ª Conferência Internacional de Engenharia de Software (ICSE'00).

MURPHY, A. L., PICCO, G. P., ROMAN, G.-C., 1999, "LIME: Linda Meets Mobility", v. 21ª Conferência Internacional de Engenharia de Software.

MURPHY, A. L., PICCO, G. P., 2004, "Using Coordination Middleware for Location-Aware Computing: A Lime Case Study", v. 6ª Conferência Internacional sobre Linguagens e Modelos de Coordenção (Coordination'04).

NOY, N. F., MCGUINNES, D. L., 2001, "Ontology Development 101: A Guide to Creating Your First Ontology", Access in 07/2005.

PELLET, 2006. In: http://www.mindswap.org/2003/pellet/.

POSNER, I. R., E BAECKER, R. M., 1992, "**How People Write Together**", v. IV, pp. 127-138, **25ª ConferênciaInternacional sobre Ciências de Sistemas**.

RANGANATHAN, A., CAMPBELL, R. H., 2003, "A Middleware for Context-Aware Agents in Ubiquitous Computing Environments". In: *ACM/IFIP/USENIX International Middleware Conference*, Rio de Janeiro, Brasil.

ROSA, M. G. P., BORGES, M. R. S., SANTORO, F. M., 2005, "Avaliando a Influência do contexto da Interação no Nível de Cooperação de um Grupo", WCSCW.

SAPSOMBOON, B., ANDRIATI, R., ROBERTS, L., SPRING, M. B., 1997, *Software to Aid Collaboration: Focus on Collaborative Authoring*. University of Pittsburgh.

SCHILIT, B., ADAMS, N., WANT, R., 1994, "Context-Aware Computing Applications". In: *Proc. of the Workshop on Mobile Computing Systems and Applications*, Santa Cruz, CA.

SCHILIT, B., THEIMER, M., 1994, "Disseminating Active Map Information to Mobile Hosts", pp. 22-32, IEEE Network.

SCHILIT, B. T. M., 1994, "Disseminating Active Map Information to Mobile Hosts", pp. 22-32, IEEE Network.

VIEIRA, V., TEDESCO, P., SALGADO, A. C., 2005a, "Towards an Ontology for Context Representation in Groupware". In: *Proceedings of the 11th International Workshop on Groupware (CRIWG'05)*, pp. 367-375, Porto de Galinhas, Brasil.

VIEIRA, V., LUCENA, B., ROCHA, F., 2005b, ***Usando Espaço de Tuplas para Criar um Editor de Escrita Colaborativa Móvel e Multi-Síncrono***.

VIEIRA, V., LUCENA, B., ROCHA, F., 2005c, "CoWS - Collaborative Writing through Shared Spaces". In: http://www.cin.ufpe.br/~vvs/cows/.

W3C, 2006, "OWL Web Ontology Language Overview".

WANG, X. H., GU, T., ZHANG, D. Q., PUNG, H. K., 2004, "Ontology based context modeling and reasoning using OWL". In: *Workshop on Context Modeling and Reasoning at II IEEE International Conference on Pervasive Computing and Communication*, Orlando, Florida.

WU., S., SIEGEL, M., ABLAY, S., 2002, "**Sensor Fusion for Context Understanding**". In: ***Proceedings of IEEEInstrumentation and Measurement Technology Conference***.

Apêndice A - glossário

Todas essas definições foram retiradas do próprio trabalho, exceto quando explicitamente referenciadas. Estes termos, bastante usados neste trabalho, são aqui explanados.

A – E

Builtin: são os chamados procedimentos básicos do Jena.

CoBrA (Context Broker Architecture): Uma arquitetura criada por Chen para o desenvolvimento de aplicações voltadas a sistemas móveis que possuem ciência de contexto. A ontologia é feita em OWL. Foca em manter um modelo contextual compartilhado entre todos os membros mantendo a privacidade de informações.

CSCW (*Computer Supported Cooperative Work*): Área que estuda meios de se melhorar tarefas cooperativas com o apoio de computadores.

DAM+OIL: linguagem feita para representar recursos, baseada em RDF. Está sendo substituída por OWL.

Framework: 1a: Uma estrutura conceitual básica para algo (por exemplo, idéias). b: um esqueleto, uma estrutura (MERRIAN WEBSTER ONLINE, 2005).

F – J

GDSS (*Group Decision Support Systems*): sistemas de groupware que dão suporte informatizado a reuniões executivas, por exemplo um sistema instalado em uma sala de reunião em cada membro com o seu terminal pode checar gráficos e estatísticas em suas maquinas, com um quadro que o apresentador escreve e todos podem ver do seu terminal. Isso seria um GDSS.

Instância: Em uma ontologia, instância é um individuo de uma classe da uma ontologia.

Jena: Framework Java para tratamento de ontologias e com suporte a máquinas de inferência.

K – O

LIME (*Linda in a Mobile Environment*): é um middleware que facilita a criação de aplicativos com mobilidade tanto física quanto lógica.

LINDA: framework da década de oitenta que propõe a criação de aplicativos baseados na idéia de espaços de tupla que são espaços de memória

compartilhados que são tratados por operações de inserção remoção e leitura dos dados.

Middleware: interface que permite interação de diferentes aplicações de softwares, possivelmente sobre diferentes plataformas de hardware e infraestrutura, para troca de dados.

OntoContext: Ontologia proposta por Vieira *et al* (VIEIRA et al., 2005a) para representação de contexto em groupware.

Ontologia: é uma estrutura de hierarquia de dados contendo todas as entidades relevantes e seus relacionamentos e regras inseridas em um domínio.

OWL: linguagem ontológica criada se baseando na liberdade de tags customizados de XML com a flexibilidade de representação de dados de RDF.

P – T

Protégé*:* Programa para criação e edição de ontologias. Atualmente na versão 3.2.

Reasoner/raciocinador: É a parte do Jena que cuida da inferência se baseando em regras definidas pelo desenvolvedor.

RDF: é um modelo de dados para objetos (recursos) e relações entre eles. possui semântica relativamente simples e pode ser representado como XML, precursor de OWL (W3C, 2006).

RDFS: RDF Schema é um vocabulário para descrever propriedades e classes de recursos RDF, com semântica suficiente para generalizar hierarquias de tais propriedades e classes.

U – Z

Web Semântica: é a visão de futuro em que toda a informação vai possuir um significado que as máquinas possam entender e automaticamente processar e integrar informação da web.

XML: provê uma sintaxe para documentos estruturados, mas não dão suporte a restrições semânticas nesses documentos.

XML Schema: linguagem que restringe a estrutura de documentos XML e estende XML com *datatypes*.

Diego Reynaldo Lira Llamoca Zárate
Orientando

Ana Carolina Salgado
Orientador

Vaninha Vieira
Co-orientador