Capítulo 2

Ambientes de Desenvolvimento de Software

Desenvolver software de qualidade assegurada, com elevada produtividade, dentro do prazo estabelecido e sem necessitar de mais recursos do que os alocados tem sido o grande desafio da Engenharia de Software. Diante dessa necessidade, cresce a demanda por ferramentas para suportar o desenvolvimento de software de maneira eficiente e que possam dar um apoio efetivo aos engenheiros de software na realização de suas tarefas.

Ambientes de Desenvolvimento de Software são sistemas que objetivam fornecer apoio ao processo de software, provendo um conjunto de ferramentas e facilidades integradas que sejam capazes de apoiar cada uma das atividades desse complexo processo.

Claramente, construir um ambiente que atenda a esses objetivos não é algo trivial. Há anos, pesquisadores, universidades e até mesmo a indústria de software vêm investindo nesse intento, com algumas tentativas frustradas (CHEN *et al.*, 1992) e outras que fizeram com que a pesquisa progredisse, incorporando novas características e evoluindo esses ambientes (BROWN *et al.*, 1992), (THOMAS *et al.*, 1992), (AMBRIOLA *et al.*, 1997).

Nesse caminho evolutivo, que se inicia com a criação das primeiras ferramentas isoladas e segue até os complexos ambientes que existem hoje, surgiram diversas linhas de pesquisa, criando novos tipos de ferramentas e ambientes e agregando novas funcionalidades aos existentes. Houve também a incorporação de inúmeras tecnologias e disciplinas que não faziam parte da Engenharia de Software, mas de áreas como Inteligência Artificial, Bancos de Dados e até Administração. Durante a história dos ambientes, diversos exemplares foram sendo criados e são citados ao longo deste capítulo, que busca contextualizar os ambientes de desenvolvimento de software e, para isso, está organizado da seguinte forma: a seção 2.1 – *Evolução de ADSs* – percorre a principal linha evolutiva desses ambientes, apresentando seus tipos, características e exemplos; a seção 2.2 – *Áreas de Pesquisa Relacionadas* – discute algumas das áreas de pesquisa mais importantes que contribuíram para o crescimento desses ambientes; a seção 2.3 – *O Ambiente ODE* – apresenta o ambiente ODE que é o foco de

aplicação da pesquisa desenvolvida neste trabalho; por fim, a seção 2.4 apresenta as considerações finais e conclusões do capítulo.

2.1. Evolução de ADSs

Ferramentas para apoiar desenvolvedores na produção de software têm existido, de uma forma ou de outra, desde os primeiros dias da programação em computadores (HARRISON *et al.*, 2000), (GRUHN, 2002). Elas estão se tornando mais necessárias à medida que a demanda por software aumenta, o tempo de desenvolvimento reduz e a diversidade e a complexidade crescem além do imaginado há poucas décadas atrás (HARRISON *et al.*, 2000).

Para atender a essa crescente demanda por ferramentas de apoio ao desenvolvimento de software, muito esforço tem sido feito no sentido de criar ferramentas que ofereçam apoio às mais diversas atividades envolvidas no processo de software e de fazer com que elas evoluam, trabalhem em conjunto e sejam realmente efetivas no propósito para o qual foram construídas (BROWN *et al.*, 1992), (AMBRIOLA *et al.*, 1997), (HARRISON *et al.*, 2000), (HOLZ *et al.*, 2001), (FISCHER *et al.*, 2001), (GRUHN, 2002), (OLIVEIRA *et al.*, 2004), (LIMA, 2004).

A primeira geração de ferramentas de apoio ao desenvolvimento de software foi concebida por volta dos anos 1970. Essas ferramentas proviam suporte isolado a tarefas individuais como edição de código, compilação, depuração etc. (GRUHN, 2002). Desde então, houve uma evolução consideravelmente rápida das ferramentas de apoio. Nesse histórico evolutivo é importante destacar algumas classes de ferramentas, tais como ferramentas CASE, Ambientes de Desenvolvimento de Software (ADSs), ADSs Centrados em Processo, ADSs Orientados a Domínio e ADSs utilizando Gerência de Conhecimento em Engenharia de Software; assim como os fatores que motivaram essa evolução e as técnicas que foram sendo empregadas ao longo do anos.

2.1.1 – Ferramentas CASE

Todo engenheiro de software utiliza ferramentas e elas têm sido criadas desde os dias dos primeiros montadores (HARRISON *et al.*, 2000). Através do tempo, o número e a variedade de ferramentas têm crescido tremendamente. Elas abrangem desde ferramentas tradicionais, como editores, compiladores e depuradores, até ferramentas que apóiam a coleta

de requisitos, análise e projeto de sistemas, construção de interfaces gráficas, geração de consultas, arquitetura de sistemas e conexão de componentes, testes, gerência de configuração, administração de bancos de dados, reegenharia, engenharia reversa, visualização de programas e coleta de métricas (HARRISON *et al.*, 2000).

Essas ferramentas, conhecidas como ferramentas CASE (*Computer Aided Software Engineering*), tornaram-se extremamente úteis, fornecendo apoio às tarefas para as quais foram construídas.

Atualmente, as ferramentas CASE são amplamente utilizadas pela indústria de software. Ferramentas comerciais apóiam a realização de grande parte das atividades que compõem o processo de software, principalmente as relacionadas à construção.

Entretanto, cada uma dessas ferramentas geralmente atende a uma, ou a poucas atividades, resolvendo apenas problemas pontuais. Isso implica na utilização de várias ferramentas para que se possa atender às diversas atividades do processo. Além disso, são raras as vezes em que conseguem comunicar-se entre si, trocando informações ou serviços.

Segundo Gruhn (GRUHN, 2002), quanto mais essas ferramentas isoladas foram sendo construídas e utilizadas, mais urgente se tornou a necessidade de integrá-las de alguma maneira. Resultados produzidos por uma ferramenta devem ser passíveis de processamento por outra ferramenta. Assim, ao menos algum tipo de integração de dados tornou-se necessária.

Durante as décadas de 1980 e 1990, um dos conceitos mais populares entre os pesquisadores de ferramentas de apoio ao desenvolvimento de software era "integração". As ferramentas de apoio a tarefas pontuais, que já ocupavam um lugar no mercado, começaram a perder espaço entre os pesquisadores. Havia um crescente interesse na pesquisa dos chamados Ambientes CASE Integrados (CHEN *et al.*, 1992).

Acreditava-se que esses ambientes tomariam o lugar de ferramentas CASE individuais, pois seriam capazes de abrigar várias ferramentas integradas em um ambiente coeso que oferecesse suporte a todo o ciclo de vida do software, permitindo um aumento de produtividade e qualidade e redução de custos (CHEN *et al.*, 1992), (THOMAS *et al.*, 1992).

Diversas outras previsões, muitas vezes utópicas, foram feitas quanto a esses ambientes. Entretanto, os ambientes integrados ainda não foram capazes de tomar o lugar das ferramentas CASE, que são largamente utilizadas pelo mercado de software atualmente.

2.1.2 – Ambientes de Desenvolvimento de Software

Com o passar do tempo, o conceito que se consolidou quanto à integração de ferramentas foi o de Ambiente de Desenvolvimento de Software (ADS) (ou, em inglês, *Software Engineering Environment* - SEE). Embora a pesquisa tenha buscado metas mais adequadas à realidade, a finalidade dos ambientes não se alterou muito (HARRISON *et al.*, 2000). Assim, basicamente, o objetivo de um ADS é prover um ambiente capaz de apoiar todo o processo de desenvolvimento de software, integrando diversas ferramentas que possam trabalhar em conjunto.

Entretanto, a simples união das ferramentas em um mesmo sistema não é o suficiente. A integração em ADSs toma um significado muito mais amplo. Cada ferramenta passa a fazer parte de um todo. Deve haver um compartilhamento em vários níveis, englobando conceitos, funcionalidades, estruturas, representações e informações. Dessa forma, ADSs buscam combinar técnicas, métodos e ferramentas para apoiar o engenheiro de software na construção de produtos de software, abrangendo todas as atividades inerentes ao processo, tais como as de gerência, desenvolvimento e controle da qualidade (FALBO, 1998).

A integração vem sendo considerada um dos mais desafiantes tópicos na pesquisa em ambientes de desenvolvimento de software. A integração demanda representação consistente da informação, interfaces padronizadas entre ferramentas, significados homogêneos da comunicação entre engenheiros de software e ferramentas, e uma efetiva abordagem que possibilite aos ADSs funcionar em várias plataformas (PRESSMAN, 2004).

Por ser um aspecto bastante amplo, a questão da integração em ADSs comumente é dividida em dimensões. Muitas dimensões de integração são tratadas na literatura, mas as que recebem maior destaque são (PFLEEGER, 2001), (FALBO *et al.*, 2004c):

- Integração de Dados: essa dimensão lida com a maneira como o ambiente e suas ferramentas compartilham dados. Essa foi, possivelmente, a primeira dimensão em que houve necessidade de integração. Não seria possível integrar ferramentas sem que estas pudessem trocar os dados que produzissem.
- Integração de Controle: permite que o ambiente perceba quando e quais funcionalidades são executadas e também que seja capaz de realizá-las quando necessário. Tem por objetivo suportar uma combinação flexível das funções do ambiente e de suas ferramentas, por meio do compartilhamento de suas funcionalidades ou serviços.

- Integração de Processo: estabelece uma ligação explícita entre as ferramentas e o processo de software seguido pelo ambiente. Permite que as ferramentas e os recursos definidos no processo se relacionem adequadamente. Tem a intenção de garantir que as ferramentas interajam efetivamente no apoio ao processo definido.
- Integração de Apresentação: mantém as interfaces do ambiente homogêneas e consistentes, permitindo que os usuários alternem entre várias ferramentas sem mudanças substanciais de estilo e aparência. Objetiva melhorar a efetividade da interação com o usuário, considerando desde ferramentas individuais até o ambiente como um todo.
- Integração de Conhecimento: permite que o conhecimento armazenado no ambiente esteja disponível às ferramentas ou usuários que precisem acessá-lo. Refere-se ao gerenciamento do conhecimento capturado durante projetos de software e à oferta de apoio baseado em conhecimento aos engenheiros de software.

Segundo (BROWN *et al.*, 1992), o nível de detalhes compartilhados pelas ferramentas é uma importante medida de integração. Na medida em que o nível de integração dessas dimensões aumenta, o ambiente se torna mais homogêneo, consistente, robusto e, por conseqüência, capaz de apoiar mais efetivamente o processo de software. Vale ressaltar que as dimensões de integração não são tratadas em ferramentas individuais ou porções do ambiente, mas no ADS como um todo, ao passo que este cresce e evolui.

Para que as dimensões possam ser trabalhadas em ADSs, é comum o uso de diversas tecnologias de apoio. Dessa forma, para integrar dados, pode-se utilizar, por exemplo, um sistema gerenciador de banco de dados ou arquivos XML; a integração de controle pode usar orientação a objetos, orientação a aspectos, uma estrutura Modelo-Visão-Controle ou agentes; linguagens de modelagem de processos são úteis à integração de processos; e a integração de conhecimento utiliza-se de sistemas baseados em conhecimento, técnicas de apoio à gerência de conhecimento, ontologias, agentes e máquinas de inferências. Dado que algumas dessas tecnologias são de grande importância para os ADSs, elas são discutidas em mais detalhes posteriormente.

Vale ainda comentar que ao longo da história dos ADSs, alguns ambientes deram maior ênfase a alguma das dimensões apresentadas, originando novos tipos de ambientes, como é o caso dos ADSs Centrados em Processo, que privilegiaram a integração de processos, e dos ADSs Orientados a Domínio e ADSs com Gerência de Conhecimento, que

focaram na integração de conhecimento. Esses tipos de ambientes são tratados nas seções seguintes.

2.1.3 – Ambientes de Desenvolvimento de Software Centrados em Processo

Produtos de software não devem ser desenvolvidos de forma *ad hoc*, mas sim seguindo um conjunto de atividades bem definido e ordenado. Esse conjunto de atividades, mais os recursos utilizados e produzidos, forma um processo de software, que envolve, ainda, um conjunto de ferramentas e técnicas para apoio à realização das atividades (PFLEEGER, 2001). Ou seja, para que o desenvolvimento de software ocorra com qualidade, é necessário que um processo de software bem definido seja seguido.

Com o objetivo de fazer com que os ADSs apóiem as atividades, segundo um processo de software estabelecido, surge uma linha de pesquisa com maior foco na integração de processos, a de ADSs Centrados em Processos (ADSCP) (ou, em inglês, *Process-centered Software Engineering Environment* - PSEE).

Os ADSCPs integram ferramentas para dar suporte ao desenvolvimento do produto de software e para apoiar a modelagem e a execução do processo de software que desenvolve esse produto (HARRISON *et al.*, 2000). Seu objetivo é oferecer apoio a diferentes tipos de processos de software, sendo parametrizável por um modelo de processo, o qual determina como o ambiente deve se comportar. Modelos de processo, usados dessa forma, podem definir quais atividades são executadas, quando e por quem, podem identificar ferramentas a serem usadas e o formato de documentos a serem criados e manipulados (GRUHN, 2002).

ADSCPs podem ser vistos como a automatização de um processo de software e têm a responsabilidade de (CHRISTIE, 1995): (i) guiar uma seqüência de atividades definida; (ii) gerenciar os produtos que estão sendo desenvolvidos; (iii) executar ferramentas necessárias para a realização das atividades; (iv) permitir comunicação entre as pessoas; (v) colher dados de métricas automaticamente; (vi) reduzir erros humanos; e (vii) prover controle do projeto à medida que este vai sendo executado.

Segundo AMBRIOLA *et al.* (1997), durante a execução do modelo de processo, um ADSCP deve prover uma variedade de serviços relacionados ao processo, tais como assistência aos desenvolvedores de software, automatização de tarefas de rotina, invocação e controle de ferramentas e garantia da realização de tarefas e práticas obrigatórias.

Além disso, ARBAOUI *et al.* (2002) propõem um conjunto de requisitos para esses ambientes, considerando os recentes avanços e necessidades. Segundo eles, um ADSCP deve:

- fornecer uma linguagem de modelagem de processo com sintaxe e semântica definidas, bem como apoio à execução do modelo de processo;
- apoiar a ordenação dinâmica das atividades do processo de software;
- apoiar processos de software distribuídos, o que implica em apoiar a integração interpessoal formal e a interoperabilidade entre ferramentas;
- apoiar a evolução do processo de software.

Vale comentar, ainda, a necessidade de apoio à evolução do processo para que o estado de execução do modelo esteja consistente com o estado de execução do processo, uma vez que uma distância expressiva entre esses estados significa que o modelo de processo não é mais capaz de influenciar a execução do processo. No entanto, é necessária a tolerância a desvios, uma vez que situações não previstas ocorrem. Nesse caso, o ambiente tem uma visão parcial do processo real, dependendo fortemente do *feedback* fornecido pelas pessoas que estão executando o processo para tornar-se consciente do desvio.

Por fim, FUGGETTA (2000) recomenda que os ADSCPs:

- não sejam intrusivos, isto é, suavemente se integrem e complementem o ambiente de desenvolvimento tradicional, automatizando de maneira efetiva apenas os fragmentos de processo que são razoáveis automatizar;
- sejam capazes de tolerar e gerenciar inconsistências e desvios, de forma a refletir a natureza criativa da atividade de desenvolvimento de software;
- informem claramente aos desenvolvedores o estado do processo de desenvolvimento de software de muitos pontos de vista diferentes;
- sejam implantados de forma incremental, de modo que a transição para a nova tecnologia seja facilitada e os riscos sejam reduzidos.

Alguns importantes exemplos de ADSCPs ao longo da história incluem: *Adele, Argo, PCTE e SPADE* (citados em (HARRISON *et al.*, 2000)), *OIKOS* e *EPOS* (citados em (AMBRIOLA *et al.*, 1997)) e *Merlin* e *MELMAC* (citados em (GRUHN, 2002)).

2.1.4 – Ambientes de Desenvolvimento de Software Orientados a Domínio

Um dos desafios do desenvolvimento de software é a correta compreensão daquilo que o sistema necessita realizar, ou seja, os requisitos do sistema. A falta de compreensão do problema pode levar ao desenvolvimento correto do produto errado, o que pode ser agravado

pelo fato da solução estar dispersa no conhecimento de vários especialistas. Ou seja, uma das grandes dificuldades no desenvolvimento de software é que, muitas vezes, os desenvolvedores não estão familiarizados com o domínio para o qual o software está sendo desenvolvido (OLIVEIRA, 1999).

Para tratar esse problema, vários grupos de pesquisa propuseram evoluir ADSs para apoiar o desenvolvimento de software considerando características peculiares do domínio (FISCHER, 1996), (FISCHER *et al.*, 2001), (OLIVEIRA, 1999), (OLIVEIRA *et al.*, 2004). Constatou-se que ADSs apóiam melhor o desenvolvimento e a manutenção de um produto de software se forem capazes de fornecer conhecimento do domínio aos desenvolvedores (LIMA *et al.*, 2002). A partir dessa constatação e das limitações dos ADSs convencionais em apoiar o aprendizado sobre um domínio de aplicações, definiram-se os Ambientes de Desenvolvimento de Software Orientados a Domínio (ADSODs) (OLIVEIRA *et al.*, 2000). Esses ambientes propõem um apoio ao entendimento do domínio para os desenvolvedores, principalmente aqueles que não têm familiaridade ou experiência em realizar trabalhos no domínio considerado. ADSODs são definidos tendo como base os tradicionais ADSs, mas incorporando um novo fator: o conhecimento de um domínio específico (OLIVEIRA *et al.*, 2004).

Um ADSOD é um ambiente que apóia o desenvolvimento de sistemas de software em um domínio específico, considerando seu conhecimento para guiar o desenvolvedor em várias tarefas do processo de software. Esta nova classe de ADSs requer duas características essenciais:

1. O conhecimento do domínio deve ser capturado, modelado e armazenado para uso no ambiente;
2. O ambiente deve suportar a disseminação e uso do conhecimento do domínio.

Para atender à primeira característica, é importante que o conhecimento de domínio seja bem definido e formalizado. E isso tem sido alcançado através do uso de ontologias. Ontologias de domínio definem os conceitos, relações, propriedades e restrições válidos para o domínio em questão. Assim, o conhecimento de domínio pode ser definido sobre uma base conceitual robusta, que facilita a sua manipulação.

A segunda característica essencial pode ser contemplada por meio do uso de algumas facilidades de gerência de conhecimento. Uma vez formalizado e introduzido no ambiente, o conhecimento de domínio pode ser gerenciado, fazendo com que sejam promovidos seu uso e sua disseminação.

Podem-se citar alguns exemplos de ADSODs desenvolvidos. No contexto da estação TABA (ROCHA *et al.*, 1990), foram construídos os ambientes CORDIS (OLIVEIRA, 1999), para domínio de cardiologia; NETUNO (GALOTA, 2000), para o domínio de acústica submarina; SIDER (CARVALHO, 2002), para o domínio siderúrgico; e INSECTA (FOURO, 2002), para o domínio de entomologia. Além desses, (FISCHER *et al.*, 2001) apresentam EDC, um ADSOD aplicado ao domínio de planejamento urbano, que provê suporte a decisões, principalmente em planejamento de transportes e desenvolvimento comunitário.

2.1.5 – Ambientes de Desenvolvimento de Software com Gerência de Conhecimento em Engenharia de Software

Desenvolvedores de software lidam de forma intensa com diferentes tipos de conhecimento ao longo dos processos de software. O conhecimento do domínio da aplicação é uma parcela do conhecimento necessário. Uma outra parcela é constituída pelo conhecimento acumulado pela organização e relevante para o contexto específico. Conhecimento sobre diretrizes e melhores práticas organizacionais, técnicas e métodos de desenvolvimento de software, além de experiências anteriores com o uso dessas técnicas e métodos e com o processo de software são exemplos de conhecimento relevante nesse contexto. No entanto, identificação, organização, armazenamento e uso de conhecimento não são tarefas triviais.

Para atender a essa necessidade, cada vez mais a gerência de conhecimento tem se tornado presente em ADSs. Um primeiro passo foi utilizar algumas de suas técnicas para gerenciar o conhecimento de domínios de aplicação (ADSOD).

Entretanto, como o foco principal dos ambientes é apoiar o desenvolvimento de software, nada mais natural que gerenciar o conhecimento de seu próprio domínio, o de Engenharia de Software. Assim, alguns ambientes passaram a incorporar conhecimento a respeito de processos, atividades, recursos, artefatos, métodos, técnicas, paradigmas, tecnologias, entre outros (FALBO, 1998), (MAURER *et al.*, 2002), (LIMA, 2004).

Nesse contexto, intimamente relacionado ao conhecimento de engenharia de software, está o conhecimento sobre a própria organização, que também abrange os recursos, processos e atividades organizacionais, além de outros tipos de conhecimento específicos de uma organização, tais como sua estrutura, seus objetivos, suas normas etc (TIWANA, 2000), (DIERKES *et al.*, 2001), (LIMA, 2004).

Três pesquisas importantes que focam em ADSs com Gerência de Conhecimento em Engenharia de Software são os ADSOrg (LIMA, 2004), originários da Estação TABA (ROCHA *et al.*, 1990), o Projeto MILOS (HOLZ *et al.*, 2001) e a Gerência de Conhecimento em ODE (NATALI *et al.*, 2003). Esse último será explorado em detalhes na seção 2.3 deste capítulo.

O conceito de ADSOrg (Ambiente de Desenvolvimento de Software Orientado a Organização) (LIMA, 2004) surgiu a partir da necessidade de uma organização gerenciar o seu próprio conhecimento e o de seu domínio. Um ADSOrg apóia a atividade de Engenharia de Software em uma organização, fornecendo conhecimento acumulado pela organização e relevante para esta atividade, ao mesmo tempo em que apóia, a partir dos projetos específicos, o aprendizado organizacional em Engenharia de Software.

Os ADSOrg representam uma evolução dos ADSODs, tendo como objetivo evitar que o conhecimento de software fique disperso ao longo da estrutura organizacional e, conseqüentemente, sujeito a dificuldades de acesso e mesmo a perdas (LIMA *et al.*, 2000). Esses ambientes pretendem apoiar o desenvolvimento e a manutenção de software tanto em organizações em que essas são as atividades principais de negócio, quanto em organizações que possuem outro tipo de negócio e nas quais o desenvolvimento e a manutenção de software são atividades de suporte.

Para apoiar o gerenciamento do conhecimento em organizações, devem-se considerar alguns requisitos para ADSOrg, dentre eles (LIMA *et al.*, 2000):

i) ter uma representação da estrutura organizacional;

ii) reter conhecimento especializado sobre desenvolvimento e manutenção de software;

iii) permitir a utilização deste conhecimento em projetos;

iv) apoiar a atualização constante do conhecimento armazenado no ambiente; e

v) facilitar a localização de especialistas da organização que podem ser úteis em um projeto.

Os requisitos (i) e (v) estão fortemente relacionados, sendo que (i) refere-se ao modelo do conteúdo e (v) à utilização desse conteúdo. Uma equipe de projeto que tem acesso à estrutura organizacional na qual está inserida é capaz de localizar mais facilmente os especialistas que podem trazer contribuições ao projeto. Isso é muito importante quando o conhecimento necessário não está disponível no ADSOrg ou quando, apesar de disponível, sua aplicação requer entendimento mais profundo.

Para atender ao requisito (ii), um ADSOrg precisa dispor de conhecimento sobre as atividades de desenvolvimento e manutenção de software que são sempre realizadas na organização independente de um projeto específico, permitindo a elaboração de um processo padrão para a organização. Além disso, o ADSOrg precisa armazenar a experiência dessa organização em Engenharia de Software. Sempre que pertinente, cada item de experiência deve ser acompanhado da identificação de especialistas internos e materiais de referência que podem fornecer algum tipo de orientação adicional.

Com relação à utilização do conhecimento disponível sobre as atividades de desenvolvimento e manutenção de software – requisito (iii) –, ela é fundamental, principalmente, nas atividades iniciais de um projeto de desenvolvimento de software, quando são elaborados a proposta de desenvolvimento e o plano de projeto. Essas atividades são pouco apoiadas por ADSs e são centradas no conhecimento e na experiência do gerente do projeto. Os ADSOrg pretendem apoiar gerentes de projeto, transformando as atividades iniciais dos projetos de desenvolvimento de software em atividades centradas no conhecimento e na experiência de seus vários gerentes de projeto ao longo do tempo.

Uma questão importante é que o conhecimento de uma organização está em constante evolução, diferente do conhecimento sobre o domínio que tende a evoluir lentamente. Experiências relevantes no desenvolvimento de software podem ser adquiridas a cada projeto, podendo conduzir ao refinamento do repositório de conhecimento. Além disso, a infra-estrutura organizacional e o processo padrão da organização também podem ser alterados. Assim, é fundamental apoiar a atualização do conhecimento armazenado em um ADSOrg – requisito (iv).

Outra pesquisa em ADS com gerência de conhecimento organizacional é o Projeto MILOS (MAURER *et al.*, 2002), que é uma parceria da Universidade de Calgary, Canadá com a Universidade de Kaiserlautern na Alemanha, e tem a finalidade de oferecer uma infra-estrutura que integre os conceitos de ADSCP e de Gerência de Conhecimento. O ambiente construído é composto por um Ambiente de Modelagem Estendida de Processos, um Ambiente de Planejamento de Projeto, um Ambiente de Controle de *Workflow* e um Assistente de Informação que armazenam e manipulam modelos genéricos de processos, planos de projeto e dados de projetos.

2.2. Áreas de Pesquisa Relacionadas

Diversas áreas de pesquisa têm sido fonte de contribuições para a evolução dos ADSs ao longo de sua história, visando a torná-los mais úteis de alguma maneira. O uso de várias tecnologias, métodos e técnicas é útil como forma de incrementar as funcionalidades dos ambientes ou trazer novas características que, por diversas vezes, apóiam sua evolução. Assim, é possível notar, em diversos ADSs, a utilização de resultados de pesquisas tanto da área de engenharia de software como de áreas afins, como inteligência artificial, banco de dados e administração, entre outras.

Esta seção discute algumas das áreas de pesquisa relacionadas a ADS, focando, principalmente, naquelas associadas à manipulação de conhecimento. Quando se trata de manipulação de conhecimento, Gerência de Conhecimento é um conceito chave e merece destaque. Mas, aliadas à Gerência de Conhecimento, muitas outras áreas são úteis ao uso de conhecimento e apoio a atividades em ADSs, tais como Ontologias, Agentes e Máquinas de Inferência.

2.2.1 – Gerência de Conhecimento

Obter sucesso em um mercado cada vez mais competitivo depende criticamente da qualidade do conhecimento que uma organização utiliza em seus processos. Em resposta a esta necessidade, a gerência de conhecimento tem sido utilizada. Segundo Benjamins (BENJAMINS *et al.*, 1998), a gerência de conhecimento não é um produto nem uma solução que organizações possam comprar prontos. É um processo que precisa ser implementado durante um período de tempo, que envolve tanto relações humanas quanto práticas de negócio e tecnologia de informação. Desta forma, a gerência de conhecimento combina ferramentas e tecnologias para prover apoio à administração de conhecimento, gerando benefícios para a organização e para seus membros.

Uma gerência de conhecimento eficiente deve ser capaz de apoiar a criação, captura e utilização dos vários tipos de conhecimento com os quais lida. As atividades básicas de gerência de conhecimento incluem: identificação, captura, adaptação, integração, disseminação, uso e manutenção do conhecimento. No núcleo dessas atividades, encontra-se uma memória corporativa ou organizacional, apoiando o reúso e o compartilhamento do conhecimento organizacional (MARKKULA, 1999).

Os objetivos de uma organização determinam o tipo de conhecimento que ela deve capturar. Inúmeros tipos de conhecimento podem ser utilizados para evitar retrabalho e melhorar a qualidade. Processos, modelos de qualidade, artefatos desenvolvidos, experiências e lições aprendidas são alguns exemplos de tipos de conhecimento reutilizáveis. Porém, para que o reúso de conhecimento seja eficiente, é necessário um armazenamento adequado desse conhecimento. Por exemplo, os itens de conhecimento gerados em um projeto devem ser adaptados a futuras necessidades de outros projetos, agregando informações que auxiliem seu reúso (BROMMÉ *et al.*, 1999). Esses objetivos podem ser alcançados através de um Sistema de Gerência de Conhecimento efetivo.

Gerência de Conhecimento em Organizações de Software

Embora muitas organizações não percebam, conhecimento sempre foi administrado por elas em algum nível e de alguma maneira (MEEHAN *et al.*, 2002). No entanto, a gerência explícita e efetiva do conhecimento tem sido apresentada como um fator chave para o sucesso das organizações nos ambientes de negócio atuais, não sendo diferente quando o negócio é o desenvolvimento de software. Segundo (KUCZA *et al.*, 2001), muitas organizações de desenvolvimento de software têm reconhecido que, para terem sucesso no futuro, precisam administrar e utilizar conhecimento de forma mais efetiva, produtiva e inovadora, envolvendo desde indivíduos e equipes de projeto, até a organização como um todo. Assim, as metas gerais da Gerência de Conhecimento podem ser formuladas como: tornar os indivíduos de uma organização conscientes dos processos de conhecimento, melhorar esses processos onde problemas tenham sido identificados e apoiá-los com os meios técnicos sempre que possível e razoável (LIMA, 2004).

PRAHALAD *et al.* (1990) argumentam que a Gerência de Conhecimento:

(i) apóia a tomada de decisão eficaz e a criação de soluções inovadoras e criativas, a redução da perda de conhecimento por saída de especialistas e o combate à repetição de erros através da exploração de experiências adquiridas em projetos anteriores,

(ii) permite que não especialistas obtenham conselho especializado quando necessário,

(iii) reduz esforço duplicado por eficientemente possibilitar a construção sobre trabalho prévio, além de apoiar a consolidação de conhecimentos em

> competências e a diminuição da curva de aprendizado de novas tecnologias, tornando a organização capaz de se adaptar rapidamente a novas oportunidades.

JURISTO *et al.* (2002) relembram, ainda, que, na verdade, o desenvolvimento de software ainda tem que atingir o nível de engenharia e que, para isso, é necessário identificar relações de causa e efeito que permitam explorar a eficácia das tecnologias de forma quantitativa e repetitiva. Segundo PFLEEGER (1999), se observarmos por tempo suficiente e com atenção suficiente, vamos achar regras racionais que nos mostrem as melhores formas de construir melhor um software. Nesse sentido, a Gerência de Conhecimento, que busca identificar o que funciona (e o que não funciona) no contexto da organização, é capaz de fornecer valioso material para a derivação de conclusões no nível de indústria e do domínio de conhecimento.

Aprendizado Organizacional

Para sobreviver, uma organização deve, ainda, estar comprometida com aprendizado. O sucesso depende fortemente de sua flexibilidade e dinamicidade, e isso somente pode ser efetivamente alcançado através do aprendizado. Contudo, aprendizado não pode ser pensado como um processo individual apenas. As organizações devem aprender com suas próprias experiências, lições aprendidas devem ser registradas e compartilhadas e o conhecimento relevante deve ser institucionalizado e reusado, prevenindo a repetição de erros (GARVIN, 2000).

Isso também vale para organizações de software, isto é, organizações de software devem se conduzir como organizações que aprendem continuamente. O desenvolvimento de software é um esforço coletivo, complexo e criativo e, para desenvolver produtos de software de qualidade, as organizações de software devem utilizar seu conhecimento organizacional de engenharia de software. Diversas atividades do processo de software podem ser melhoradas por meio da oferta de facilidades de gerência de conhecimento para apoiar sua realização. Algumas dessas atividades são: alocação de recursos, planejamento da qualidade, especificação de requisitos, definição de processos e gerência de riscos (FALBO *et al.*, 2004b).

2.2.2 – Ontologias

Ontologia é um termo usado para se referenciar uma compreensão compartilhada de algum domínio de interesse, que pode ser usada como uma estrutura unificada para solucionar problemas nesse domínio. Uma ontologia tenta resolver problemas como falta de comunicação dentro das organizações, o que pode gerar, por exemplo, dificuldades na identificação dos requisitos de um sistema. Ontologias consistem de conceitos e relações – o vocabulário – e suas definições, propriedades e restrições descritas na forma de axiomas (FALBO, 1998). Por meio de ontologias, é possível conseguir uma uniformidade de vocabulário, de forma a evitar ambigüidades e inconsistências. Mais precisamente, não é o vocabulário que especifica uma ontologia, mas as conceituações que os termos do vocabulário pretendem capturar (CHANDRASEKARAN *et al.*, 1999).

Acesso comum à informação é essencial para melhorar a comunicação entre desenvolvedores e ferramentas em um ADS, evitando problemas de interpretação. Especialmente em ADSs, ontologias podem ser usadas para reduzir confusões terminológicas e conceituais, facilitando o entendimento compartilhado e a comunicação entre pessoas com diferentes necessidades e pontos de vista. Além disso, a padronização de conceitos provida por uma ontologia permite que a comunicação entre as ferramentas que compõem um ambiente seja aprimorada.

No contexto da gerência de conhecimento, as ontologias podem ser vistas como a "cola" que mantém ligadas as atividades de gerência de conhecimento (STAAB, 2001). As ontologias definem um vocabulário comum utilizado pelo sistema de gerência de conhecimento e facilitam a comunicação, integração, busca, armazenamento e representação do conhecimento (O'LEARY, 1998).

2.2.3 – Agentes

Segundo (WOOLDRIDGE *et al.*, 2000), um agente é um sistema de computador que está situado em algum *ambiente* e que é capaz de executar *ações autônomas* de forma *flexível* neste ambiente, a fim de satisfazer seus objetivos de projeto. Estar situado em um ambiente significa que o agente é capaz de perceber o ambiente onde está inserido e executar ações que mudam esse ambiente de alguma forma.

No contexto da gerência de conhecimento, sistemas multi-agentes podem ser utilizados para disponibilizar o conhecimento adquirido ao longo de vários projetos para os engenheiros de software (PEZZIN, 2004).

Agentes de software podem ser utilizados para ligar os membros de uma organização ao conhecimento disponível (O'LEARY, 1998). Eles podem apoiar não só a busca e filtro, mas também, a disseminação do conhecimento. Se um processo de software tiver sido definido, agentes podem agir de forma pró-ativa, buscando e oferecendo itens de conhecimento que podem ser relevantes para a tarefa que o usuário está executando (NATALI, 2003).

Agentes podem não só recuperar os produtos de atividades executadas em projetos similares, mas, também recuperar o conhecimento informal existente no sistema sob a forma de lições aprendidas. Essas lições podem indicar pontos positivos de se realizar determinada ação e oportunidades de melhoria. Desta forma, com o apoio de agentes disponibilizando conhecimento já adquirido, as tarefas executadas por um engenheiro de software têm sua complexidade reduzida.

A seção 2.3 aborda a questão da incorporação de agentes em um ambiente de desenvolvimento de software, o ambiente ODE.

2.2.4 – Máquinas de Inferência

Uma tecnologia interessante capaz de fazer com que os sistemas executem suas funcionalidades, ou forneçam algum tipo de apoio, de forma mais inteligente são as máquinas de inferência. Através de capacidades de inferência, um sistema pode utilizar deduções lógicas para chegar a conclusões sobre as tarefas realizadas e, então, fornecer apoio mais efetivo ao usuário. Em ADSs, capacidades de inferência são especialmente úteis para permitir uma manipulação mais inteligente de conhecimento (RUY *et al.*, 2004).

Através de regras definidas e de um conhecimento prévio, um sistema pode utilizar uma máquina de inferência integrada para derivar novo conhecimento, bastante útil a tarefas complexas (RASMUS, 1995), como as envolvidas no desenvolvimento de software e em ADSs.

A definição de regras para inferência é especialmente interessante se essas regras forem definidas a partir de uma base conceitual robusta, como as ontologias. Os axiomas de uma ontologia podem dar origem a regras que manipulam o conhecimento definido com base na mesma ontologia (RUY *et al.*, 2004). Dessa forma, o novo conhecimento gerado a partir de dedução lógica possui força conceitual suficiente para servir de apoio ao desenvolvedor na realização de suas tarefas ao longo do processo de software.

A utilização de máquinas de inferência torna-se ainda mais interessante, quando aliada a agentes ou à gerência de conhecimento, conforme discutido no exemplo da próxima seção.

2.3. O Ambiente ODE

Um ADS especialmente importante no contexto deste trabalho é o Ambiente ODE, uma vez que este é utilizado como foco principal de pesquisa e experimentação dos estudos realizados.

ODE (*Ontology-based software Development Environment*) (FALBO *et al.*, 2003) é um ADS Centrado em Processo, que realiza Gerência de Conhecimento em Engenharia de Software, fundamentando-se especialmente em sua base ontológica.

As pesquisas relacionadas a ODE tiveram início em 1999, mas o ambiente só passou a existir como um ADS integrado a partir de 2002. Ao longo de sua trajetória, ODE sofreu várias evoluções e, em meados de 2004, em uma parceria universidade-empresa, o ambiente foi implantado em uma organização de software, visando a apontar oportunidades de melhoria nas ferramentas do ambiente, tomando por base situações reais dessa organização.

O ambiente é desenvolvido no Laboratório de Engenharia de Software da Universidade Federal do Espírito Santo (LabES / UFES) e utiliza em sua construção produtos de software livres, incluindo a linguagem Java e o sistema gerenciador de bancos de dados PostgreSQL, rodando no sistema operacional Linux.

ODE possui várias ferramentas, dentre elas as de apoio a: definição de processos de software (BERTOLLO *et al.*, 2006), acompanhamento de projetos (ControlPro) (DAL MORO *et al.*, 2005), gerência de recursos humanos (GerênciaRH), realização de estimativas (EstimaODE) (CARVALHO *et al.*, 2006), gerência de riscos (GeRis) (FALBO *et al.*, 2004b), documentação (XMLDoc) (NUNES *et al.*, 2004), modelagem orientada a objetos (OODE), realização de inferências (RUY *et al.*, 2004) e edição de ontologias (ODEd) (MIAN *et al.*, 2003).

Além disso, há outras pesquisas relacionadas ao ambiente ODE que merecem destaque no contexto deste trabalho. São elas:

i) sua base ontológica;

ii) sua infra-estrutura de gerência de conhecimento;

iii) sua infra-estrutura de construção de agentes; e

iv) sua infra-estrutura de inferência.

Base Ontológica

Em ODE, parte-se do pressuposto que, se as ferramentas de um ADS são construídas baseadas em ontologias, a integração delas pode ser facilitada, pois os conceitos envolvidos são bem definidos pelas ontologias (FALBO *et al.*, 2003).

Dentre as ontologias que compõem a base ontológica de ODE, têm-se as ontologias de processo de software (FALBO, 1998), (BERTOLLO, 2006), de qualidade de software (DUARTE, 2001), de artefatos de software (NUNES, 2005), de gerência de configuração de software (NUNES, 2005), de riscos de software (FALBO *et al.*, 2004b) e de requisitos (NARDI *et al.*, 2006).

A instanciação dessas ontologias dá origem a uma parte importante do conhecimento do ambiente. Esse conhecimento é usado para apoiar o usuário em diversas tarefas, tais como definição de processos, alocação de recursos, avaliação de qualidade, gerenciamento de riscos, levantamento de requisitos etc. Além disso, as ontologias são utilizadas para estruturar o ambiente e sua infra-estrutura de gerência de conhecimento, para estabelecer uma forma padrão de comunicação entre os agentes que atuam no ambiente e como base para a realização de inferências.

Infra-Estrutura de Gerência de Conhecimento

Um trabalho importante desenvolvido no contexto de ODE é a sua Infra-estrutura de Gerência de Conhecimento (NATALI, 2003). Essa infra-estrutura integra um Sistema de Gerência de Conhecimento a ODE, com o objetivo de apoiar a administração de parte do conhecimento gerado durante o processo de software.

Essa infra-estrutura foi desenvolvida de modo que haja uma memória organizacional ao centro, cercada por um conjunto de serviços de gerência de conhecimento, como mostra a Figura 2.1, que incluem:

- Criação e Captura de Conhecimento: responsável por oferecer mecanismos para obtenção e armazenamento do conhecimento;
- Recuperação e Acesso ao Conhecimento: responsável por oferecer mecanismos de busca dos itens de conhecimento armazenados na memória organizacional;

- Disseminação de Conhecimento: serviço pró-ativo realizado por agentes de software com o intuito de disponibilizar aos usuários itens de conhecimento potencialmente úteis a uma dada tarefa.
- Uso do Conhecimento: responsável por apoiar o reúso, por parte do usuário, do conhecimento existente e oferecer mecanismos de realimentação sobre a utilidade do conhecimento apresentado; e
- Manutenção do Conhecimento: responsável pelo gerenciamento dos repositórios de conhecimento, tomando por base o *feedback* dos usuários.

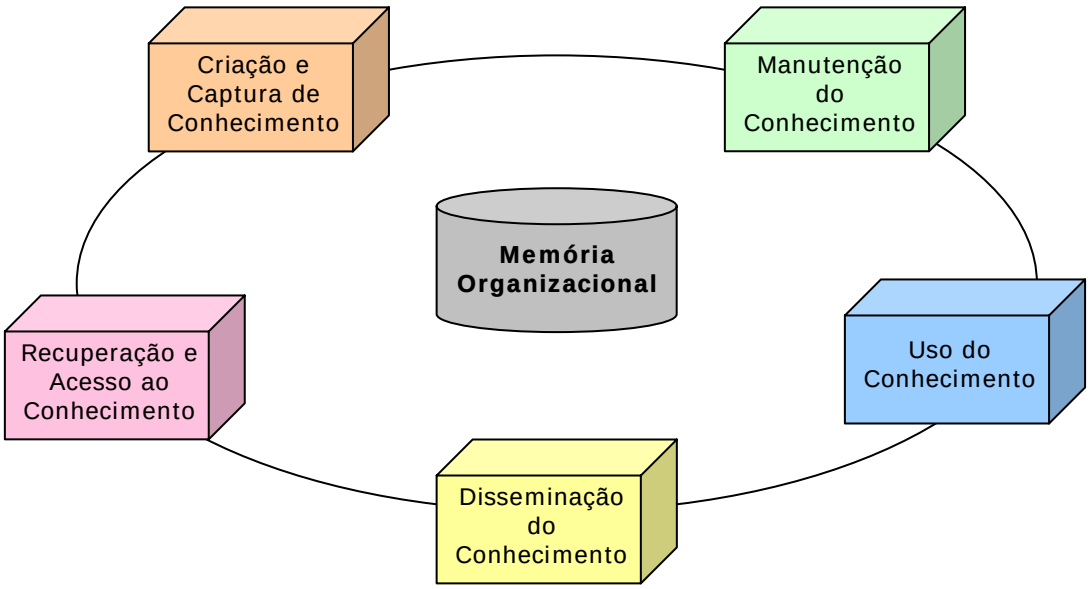

Figura 2.1 – Infra-Estrutura de Gerência de Conhecimento de ODE (NATALI, 2003)

Conforme apontado anteriormente neste capítulo, ontologias definem um vocabulário comum a ser utilizado pelo sistema de gerência de conhecimento e, por conseguinte, facilitam a comunicação, integração, busca, armazenamento e representação do conhecimento. Com base nessa premissa, a estrutura da memória organizacional de ODE é definida, também, fortemente apoiada em ontologias.

Na memória organizacional, os itens de conhecimento podem ser classificados em itens de conhecimento formais e informais. Os itens de conhecimento formais são os diversos tipos de artefatos gerados pelas ferramentas do ambiente, tais como planos de riscos, estimativas, processos e alocações de recursos. Já os itens de conhecimento informais compreendem, atualmente, lições aprendidas e pacotes de mensagens.

Infra-Estrutura de Construção de Agentes

Por serem sistemas de software complexos, ADSs são potenciais beneficiários da tecnologia de agentes. Como um exemplo da utilidade dessa tecnologia, pode-se citar a disseminação de conhecimento na gerência de conhecimento. Assim, em ODE, agentes são utilizados para aperfeiçoar algumas das funcionalidades do ambiente e uma infra-estrutura para apoiar a construção de agentes para atuarem em ODE, denominada AgeODE (PEZZIN *et al.*, 2004), foi desenvolvida.

Agentes devem ter a habilidade de se comunicar. Essa habilidade é parte percepção (o recebimento de mensagens) e parte ação (o envio de mensagens). A comunicação entre os agentes de ODE é feita usando KQML (*Knowledge Query and Manipulation Language*) (FININ *et al.* 1995). As primitivas de KQML definem as ações admissíveis que os agentes podem tentar na comunicação com outros agentes.

Novamente, ontologias são bastante úteis, agora para apoiar a comunicação entre os agentes. O projeto de um sistema multiagente requer a definição de um modelo do ambiente no qual o agente atua, para que este possa conversar sobre ele. Esse modelo pode ser exatamente uma ontologia. Assim, para que um agente consiga se comunicar com outro agente, ambos devem conhecer a(s) mesma(s) ontologia(s) (PEZZIN *et al.*, 2004).

Dessa forma, AgeODE dotou o ambiente de uma base sobre a qual novos agentes podem ser construídos seguindo o padrão criado e podendo comunicar-se com os agentes já existentes. Esses agentes são capazes de aumentar o potencial do ambiente quando atuam para apoiar a gerência de conhecimento, alocação de recursos, definição de processos, configuração automática do ambiente, identificação e avaliação de riscos etc.

Infra-estrutura de Inferência

Outra pesquisa interessante relacionada a ODE é a sua Infra-estrutura de Inferência (RUY, 2003), desenvolvida para que o ambiente possa manipular conhecimento de uma maneira alternativa, segundo o paradigma lógico.

O funcionamento da infra-estrutura consiste de três passos principais que são:

i) criação de modelos de base de conhecimento, que utiliza um editor para montar os conceitos e regras das bases de conhecimento;

ii) instanciação das bases de conhecimento por meio de um processo de extração de dados do ambiente para criar instâncias dos conceitos modelados; e

iii) realização das inferências, utilizando as bases de conhecimento em uma máquina Prolog para derivar as conclusões lógicas desejadas a respeito do conhecimento que está sendo manipulado. Por fim, os resultados são transformados em objetos para servir de apoio a diversas tarefas do ambiente.

Uma maior efetividade é alcançada se os conceitos e regras modelados são espelhados em ontologias do ambiente, uma vez que o conhecimento manipulado possui uma ligação semântica mais forte com os conceitos nos quais o ambiente se baseia.

Assim como as pesquisas mencionadas anteriormente, esta também pode ser associada às demais, oferecendo um suporte mais inteligente aos usuários de ODE.

Como é possível perceber, ODE está estruturado sobre uma robusta base ontológica. Nessa base, apóiam-se também diversas pesquisas e trabalhos que podem funcionar isoladamente, oferecendo facilidades de gerência de conhecimento, agentes, ou capacidades de inferências às ferramentas do ambiente, ou podem complementar-se, provendo um conjunto de facilidades mais amplo e efetivo.

2.4. Conclusões do Capítulo

Este capítulo discute algumas das crescentes necessidades de automatização e suporte inteligente existentes na área de engenharia de software. Contempla também algumas das soluções pesquisadas e desenvolvidas pela própria área de engenharia de software, abordando as ferramentas de apoio, seguindo pela linha de evolução dos Ambientes de Desenvolvimento de Software e instrumentos de suporte utilizados.

O desejo de criar sistemas inteligentes, capazes de realizar tarefas que não sejam meramente mecânicas, é bastante antigo. No entanto, a realização deste ato ocorre de forma lenta e gradativa. Mas, com o passar dos anos e o avanço das pesquisas, é possível aproximar-se deste intento.

Na engenharia de software, uma área em que conhecimento e experiência são características fundamentais, o foco é a automatização de tarefas complexas. Essas tarefas necessitam de conhecimento, seja da própria área de engenharia de software, de domínios de aplicação, conhecimento organizacional, ou mesmo experiência de engenheiros de software.

No contexto de ADSs, essa automatização baseada em conhecimento é caracterizada desde as primeiras tentativas de integração em que o compartilhamento de dados entre as

ferramentas tornava-se algo necessário para reduzir o esforço dos engenheiros de software. A partir de então, outros avanços podem ser considerados, como a troca de dados com algum significado embutido; a incorporação de conhecimento de diversos tipos aos ambientes; e a manipulação deste conhecimento, seja de forma direta, ou apoiada por modelos conceituais robustos. Enfim, um desejo dos ambientes atuais é estarem aptos a adquirir conhecimento, entender seu significado, manipulá-lo de forma inteligente e prover ao usuário apoio efetivo às suas tarefas.

Um trabalho que merece ser lembrado nesse campo é (BROWN *et al.*, 1992), em que, no início da década de 1990, os autores já discutiam um "nível semântico". Segundo eles, "na integração em nível semântico, as ferramentas concordam com as definições das estruturas de dados, assim como com os significados das operações sobre essas estruturas".

A parte mais interessante é que, mesmo em um momento em que os ADSs ainda davam os primeiros passos, em que a integração ocorria de forma primitiva e julgava-se ser mal entendida por muitos, o trabalho já visava um nível de integração em que as estruturas e operações tinham seu significado entendido pelo ambiente.

Desde aquela época, muito se avançou em ADSs. E, em um momento em que é possível unir em um único ambiente tecnologias como as de apoio à gerência de conhecimento, ontologias, meta-dados, técnicas avançadas de inteligência artificial e várias outras, o entendimento por parte do ambiente pode assumir proporções muito maiores, aproximando-se do que (BROWN *et al.*, 1992) apontavam como o "nível semântico".