COMPOR - Desenvolvimento de Software para Sistemas Multiagentes

Hyggo Oliveira de Almeida

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Informática da Universidade Federal de Campina Grande como parte dos requisitos necessários para obtenção do grau de Mestre em Informática.

Área de Concentração: Ciência da computação

Linha de Pesquisa: Modelos Computacionais e cognitivos

Evandro de Barros Costa

Angelo Perkusich

(Orientadores)

Campina Grande, Paraíba, Brasil

Resumo

A abordagem multiagentes tem sido apontada por muitos pesquisadores como adequada para o desenvolvimento de sistemas de software complexos. Investimentos em Engenharia de Software para prover ferramentas e metodologias para a utilização desta abordagem em larga escala têm sido realizados. Porém, nestes trabalhos não são levadas em conta características como dinamicidade do sistema e flexibilidade e adaptabilidade da solução. Neste trabalho, apresenta-se uma infra-estrutura de engenharia para o desenvolvimento de software utilizando a abordagem multiagentes denominada COMPOR. Trata-se de um conjunto de diretrizes e ferramentas para o desenvolvimento de software para sistemas complexos, abertos e dinâmicos, com foco na obtenção de flexibilidade e adaptabilidade da solução. Para ilustrar a aplicabilidade da infra-estrutura, um exemplo de desenvolvimento de um sistema complexo no domínio de empresas virtuais é apresentado.

Abstract

Many researchers have been claimed that the multiagent approach is suitable for the development of complex software systems. Many works have been proposed focusing methodologies and tools to apply the multiagent approach on large scale software development. However, the dynamic feature of the system and the flexibility and adaptability of the solution have not been considered. This work presents an engineering infrastructure to software development through a multiagent approach, named COMPOR. This infrastructure represents a set of guidelines and tools to develop software for complex, open and dynamic systems. Flexibility and adaptability of the provided solution are the main goal of COMPOR. In order to illustrate its usefulness, the development of a complex virtual enterprise application is presented.

Agradecimentos

A Deus, por sempre me proporcionar mais do que realmente mereço.

Aos meus pais e irmão, por me apoiarem em todos os momentos da minha vida, sendo o mestrado apenas mais um destes momentos.

Aos meus orientadores Evandro e Angelo, pela liberdade que me foi dada para expor e por em prática as minhas idéias, pelo incentivo e apoio técnico na realização deste trabalho e pelas proveitosas conversas que tornaram válido, verdadeiramente, meu mestrado.

A meus amigos orientandos Rodrigo Paes, Glauber Vinícius e Emerson Loureiro, com os quais tive a oportunidade de aprender muito mais que orientar e sem os quais este trabalho não seria possível.

Aos colegas do laboratório de pesquisas do TCI/UFAL, por tornarem prazeroso o trabalho do dia a dia.

Aos meus amigos de copo, indispensáveis à sobrevivência do meu bom humor.

À minha namorada Clara, pelo apoio, compreensão e, acima de tudo, paciência perante minha egoísta mania de trabalhar.

À CAPES pelo apoio financeiro.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

No mundo real, existem várias categorias de problemas que exibem características encontradas em sistemas com comportamento complexo [Jos02]. Sistemas complexos podem ser definidos como sistemas cuja execução depende da interação entre as entidades que o compõem. Exemplos típicos de sistemas complexos podem ser encontrados em sistemas biológicos, econômicos, políticos, sociais e cognitivos. De um ponto de vista computacional, muitos dos problemas atualmente tratados pela Engenharia de Software vêm se mostrando cada vez mais complexos e as entidades de abstração utilizadas por esta engenharia acompanham esta evolução. Muitos pesquisadores têm apontado a abordagem multiagentes como adequada ao desenvolvimento de sistemas de software complexos [Woo98; ZP04; Jen01; GL02], utilizando uma organização de agentes autônomos e inteligentes como abstração para gerenciar a complexidade dos sistemas.

Embora a Engenharia de Software baseada em Agentes tenha como promessa fornecer aos desenvolvedores o poder de expressão e de computação para lidar com problemas de alta complexidade, são necessárias metodologias, ferramentas e infra-estruturas gerenciáveis para permitir a construção em larga escala de sistemas multiagentes [TEC03].

1.1 Contexto de pesquisa

Este trabalho se situa na área de Engenharia de Software, tendo seu foco no desenvolvimento de sistemas de software complexos, utilizando uma abordagem multiagentes. Os primeiros investimentos na área de Engenharia de Software para sistemas multiagentes, que serviram

como base conceitual e empírica para esta dissertação, tiveram início com a tese de doutorado de Evandro de Barros Costa, orientador deste trabalho, em 1997, e a aprovação do Projeto Math_Net [Cos99]. Em tal tese propõe-se um arcabouço conceitual para a concepção de Sistemas Tutores Inteligentes(STI's) [Sel99] baseados em uma abordagem multiagentes, denominado MATHEMA [Cos97].

O Projeto Math_Net, como uma proposta de extensão do MATHEMA, teve como objetivo central o desenvolvimento de um Sistema Tutor Inteligente Multiagentes para tratar um processo de ensino-aprendizagem, realizado através da interação entre grupos de alunos, professor e sistema computacional. Resultados do MATHEMA foram ainda utilizados no contexto do Projeto MaAE [Gir01], dando uma ênfase mais forte a aspectos de Engenharia de Software. No contexto do MATHEMA e do Math_Net, foram realizados experimentos que culminaram com o desenvolvimento de sistemas tutores inteligentes em domínios, tais como Álgebra [CLF95], Lógica [CPF98], Redes de Petri [CGFP98] e Harmonia musical [CAL$^+$01; CAL$^+$02; CFA02].

Após a aplicação do modelo MATHEMA nestes domínios, foram identificados padrões de implementação e arquitetura destes sistemas, o que resultou na proposição de um arcabouço para a concepção de STI's Multiagentes [CASP02] e, posteriormente, em alguns investimentos na direção de uma metodologia para o desenvolvimento de tais sistemas [CAS$^+$02].

Durante a definição do arcabouço e de diretrizes metodológicas para o desenvolvimento de STI's Multiagentes, constatou-se que o modelo conceitual MATHEMA se adequava à concepção de sistemas multiagentes de forma geral, ou seja, em domínios de aplicação diferentes. Esta constatação encorajou a pesquisa em Engenharia de Software para sistemas multiagentes, que culminou na proposição de um novo caminho para a construção de tais sistemas.

1.2 Problemática

Muitos investimentos em Engenharia de Software para prover ferramentas e metodologias para a utilização da abordagem multiagentes em larga escala têm sido realizados. Porém, em boa parte destes trabalhos não é levada em conta a característica de dinamicidade requerida

em sistema multiagentes. Sendo assim, os relacionamentos entre os agentes do sistema não se alteram e novos agentes não podem passar a fazer parte do sistema. Em resumo, o sistema não reflete a realidade de cenários evolutivos.

Em se tratando de software, os requisitos definidos em primeira instância de análise tendem a ser refinados com a iteração do ciclo de desenvolvimento. Em relação à abstração multiagentes, isto se reflete na alteração dos relacionamentos dos agentes e na entrada ou saída de agentes no sistema. Uma vez que não se considera a dinâmica do sistema multiagentes, não são considerados também os prováveis refinamentos nos requisitos que deram origem ao sistema.

Uma infra-estrutura de engenharia para lidar com sistemas complexos deve possibilitar esta dinamicidade do sistema multiagentes como suporte à gerência da complexidade. Quanto mais flexível for a solução multiagentes, mais adaptável esta será diante de refinamentos dos requisitos.

Além disso, diretrizes para o desenvolvimento de software para sistemas multiagentes devem guiar o desenvolvedor desde à análise do problema até à implementação do software. Muitos trabalhos que propõem diretrizes para o desenvolvimento baseado em agentes levam em conta apenas as fases de análise e projeto, não considerando questões de implementação. Este talvez seja o motivo pelo qual a abordagem de agentes ainda não é utilizada em larga escala.

1.3 Objetivos da dissertação

O objetivo principal deste trabalho é disponibilizar uma infra-estrutura de engenharia para o desenvolvimento de software para sistemas complexos, abertos e dinâmicos, através de uma abordagem multiagentes. Trata-se da intenção de propor um conjunto de diretrizes e ferramentas para o desenvolvimento de software para sistemas complexos, abertos e dinâmicos, com foco na obtenção de flexibilidade e adaptabilidade [1] da solução. Estas diretrizes devem guiar o desenvolvedor desde a análise do problema até a implementação do software, sendo auxiliado pelas ferramentas.

[1] No contexto deste trabalho, adaptabilidade diz respeito à facilidade de alteração no sistema diante de cenários evolutivos. Características de auto-adaptação a mudanças não são consideradas.

Mais especificamente, pode-se dividir este objetivo principal nos objetivos específicos descritos a seguir.

- Definir um conjunto de diretrizes de análise, projeto, implementação e testes para o desenvolvimento de software para sistemas multiagentes levando em conta as características de dinamicidade dos sistemas multiagentes.

- Definir um modelo de componentes para estabelecer regras sobre a composição de funcionalidades dos agentes. O modelo deve permitir a troca de funcionalidades e componentes com o mínimo de impacto sobre o restante do agente, garantindo assim uma maior flexibilidade na solução proposta.

- Desenvolver um arcabouço baseado em componentes que implemente as especificações do modelo de componentes definido anteriormente.

- Desenvolver um ambiente integrado para a construção de software para sistemas multiagentes utilizando como base o arcabouço definido anteriormente. O ambiente também deve contemplar ferramentas para monitoramento e execução do software desenvolvido.

- Desenvolver um sistema considerado complexo, justificar sua complexidade e apresentar a aplicabilidade da infra-estrutura definida anteriormente neste sistema, como um estudo de caso.

1.4 Relevância do tema

A abordagem multiagentes para o desenvolvimento de software tem sido aplicada a muitos domínios tais como Sistemas tutores inteligentes [CAL+01; CAL+02], Mineração de dados [ZAH03; KLM02], Gestão de conhecimento [HSAG03a], Recuperação de informação [FG02], Interfaces adaptativas [HSAG03b], Robótica [LW01], Simulação [CGAP02], Web semântica [PSS02], E-business [Klu00], Planejamento de viagens [CMB00], Comércio eletrônico [FCA03], Redes de sensores distribuídos [LOT03], *Health Care Management* [HJF95], dentre muitos outros.

Apesar da enorme quantidade de aplicações que se utilizam da abordagem multiagentes, ainda é necessária uma infra-estrutura adequada para o desenvolvimento destes sistemas. A maioria destas soluções foram concebidas de forma *ad hoc*, apesar dos vários trabalhos relacionados a metodologias e ferramentas para o desenvolvimento baseado em agentes.

A disponibilização de uma infra-estrutura adequada para o desenvolvimento de software para sistemas multiagentes tem aplicação direta nos domínios acima citados. Portanto, o trabalho aqui proposto, uma vez inserido neste contexto, tem sua relevância confirmada na necessidade desta infra-estrutura de engenharia para lidar com diversos domínios de problemas do mundo real.

1.5 Estrutura da dissertação

- No Capítulo 2, são apresentadas justificativas da abordagem multiagentes para o desenvolvimento de software e as noções conceituais sobre Engenharia de Software para sistemas multiagentes necessárias ao entendimento do restante do trabalho.

- No Capítulo 3, são apresentadas as diretrizes metodológicas para o desenvolvimento de sistemas multiagentes. São descritas diretrizes para análise, projeto, implementação e testes destes sistemas, assim como, a utilização da notação apresentada no Apêndice A para documentar e descrever os artefatos gerados em cada fase.

- No Capítulo 4, é apresentado o modelo de componentes que dá suporte à arquitetura de agentes COMPOR. O modelo de interação entre os componentes é descrito e discutido em relação a outras abordagens e o modelo de especificação dos componentes é apresentado.

- No Capítulo 5, apresenta-se o arcabouço de software que implementa as especificações do modelo de componentes apresentado no Capítulo 4. Além disso, apresenta-se um exemplo ilustrativo da construção de uma aplicação multiagentes utilizando o arcabouço definido.

- No Capítulo 6, apresenta-se o ambiente de desenvolvimento de sistemas multiagentes baseado nas diretrizes, modelo de componentes e arcabouço descritos nos capítulos

anteriores. Neste sentido, são descritas as ferramentas que compõem o ambiente, suas funcionalidades e arquitetura.

- No Capítulo 7, é apresentado um estudo de caso de aplicação da infra-estrutura de software apresentada neste trabalho para a concepção de um sistema no domínio de empresas virtuais.
- Por fim, no Capítulo 8, são apresentadas as conclusões deste trabalho, assim como suas perspectivas futuras.

Além disso, no Apêndice A é apresentada a notação utilizada para a documentação e descrição dos artefatos gerados durante a aplicação das diretrizes de desenvolvimento apresentadas no Capítulo 3. Nos Apêndices B, C e D é apresentado o estado da arte nas áreas relacionadas a este trabalho, destacando as vantagens e desvantagens em relação à infra-estrutura aqui proposta e à abordagem multiagentes.

- No Apêndice B é apresentada uma análise de trabalhos relacionados às diretrizes metodológicas propostas neste trabalho.
- No Apêndice C é apresentada uma análise de arcabouços correlatos ao arcabouço proposto neste trabalho.
- Finalmente, no Apêndice D é apresentada uma análise de ferramentas e ambientes correlatos ao ambiente de desenvolvimento proposto neste trabalho.

Capítulo 2

Engenharia de Software para Sistemas Multiagentes - ESSMA

Antes de apresentar as diretrizes e ferramentas COMPOR como adequadas ao desenvolvimento de sistemas de software com base em uma abordagem multiagentes, é necessário esclarecer porque investir nesta abordagem. Muitos pesquisadores [ZP04; Woo98; Jen01] têm apontado este caminho como promissor para o desenvolvimento de sistemas de software complexos, porém nestes trabalhos não são apresentados argumentos concretos para dar suporte a esta afirmação.

Neste capítulo é apresentada uma caracterização de sistemas complexos e a relação destes sistemas com a abordagem multiagentes. Esta caracterização serve como base para as justificativas apresentadas para a utilização da abordagem multiagentes no desenvolvimento de sistemas de software complexos [APCP04]. Além disso, apresenta-se, em meio às justificativas, a conceitualização relacionada à Engenharia de Software para sistemas multiagentes necessária ao entendimento do restante do trabalho.

2.1 Sistemas complexos e Sistemas Multiagentes

Um sistema é definido como um conjunto de componentes inter-relacionados e inter-dependentes, vistos de forma integrada, com propriedades não observadas individualmente [Lud76]. A complexidade destes sistemas aumenta na medida em que estes componentes representam subsistemas, caracterizando o que a literatura freqüentemente denomina

de Sistema Complexo [Jen99; Joe02; Jos02]. Sistemas de larga escala, devido ao grande número de subsistemas envolvidos, são exemplos deste tipo de sistema [Joe02].

Apesar de não haver consenso em relação à definição do termo "Agente", alguns atributos são encontrados em diversas definições da literatura [Woo98; Fer99; Jen99], tais como Autonomia, Habilidade social e Racionalidade.

- **Autonomia** - independente dos demais agentes, em termos de execução e decisão.
- **Racionalidade** - possui um objetivo e busca alcançá-lo [Jen99].
- **Habilidade social** - interage com outros agentes em prol do seu objetivo.

Estes três atributos consensuais de um agente podem ser relacionados a características de um subsistema, o qual possui, respectivamente, um comportamento autônomo, habilidade de interação com outros subsistemas e um objetivo.

Além destas propriedades, agentes estão vinculados a um domínio de conhecimento, em alguns casos representado por uma Ontologia [DFH96], e desempenham sua habilidade social através de protocolos de interação [ON00] e linguagens de comunicação de alto nível [LFP99]. A interação entre estes agentes, regidas por estas linguagens e protocolos, caracterizam organizações de agentes. O conjunto de organizações forma uma sociedade. Esta sociedade de agentes caracteriza um Sistema Multiagentes [Fer99; Jen99] [1].

Considerando cada agente como um subsistema, por exemplo [2], um sistema complexo, por definição, pode ser visto como um sistema multiagentes, pois ambos são caracterizados por interações entre as entidades autônomas que os compõem - subsistemas e agentes. Por este motivo, a abordagem multiagentes é apontada como adequada ao desenvolvimento de sistemas complexos, pois foi concebida para lidar com suas características.

No desenvolvimento de software, tem-se então no espaço de problema a descrição do sistema complexo e no espaço de solução o sistema multiagentes, como uma abstração para o espaço de problema. Na Figura 2.1 é apresentado o mapeamento da definição de sistemas complexos pertencente ao espaço do problema para a definição de sistemas multiagentes,

[1] Os termos Sociedade de agentes e Sistema multiagentes serão usados indiferentemente nesta dissertação

[2] Um subsistema pode ser decomposto em outros subsistemas dependendo de sua complexidade, da mesma forma o agente seria decomposto em agentes referentes a um nível atômico de subsistema.

pertencente ao espaço de solução. As entidades do espaço de problema(subsistemas) são mapeadas nas entidades do espaço de solução(agentes).

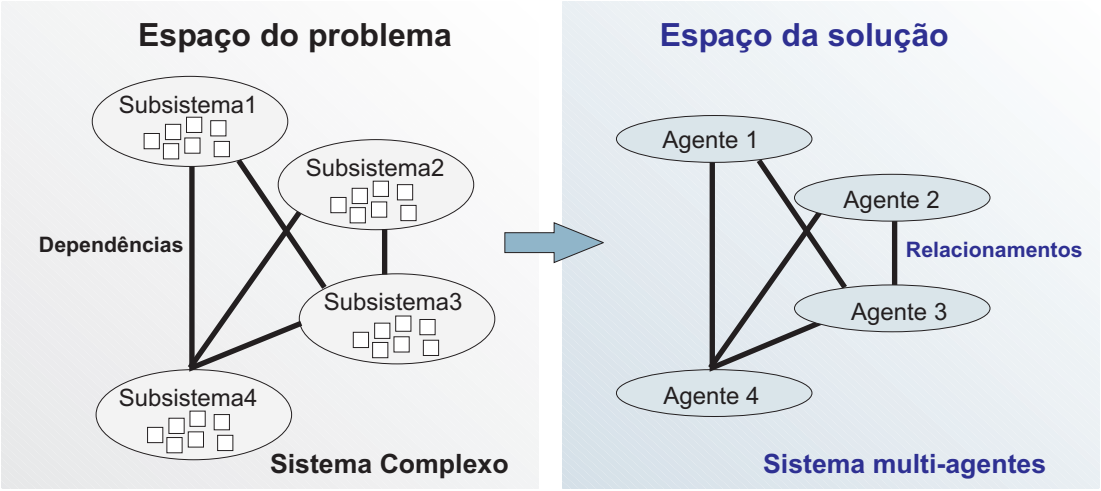

Figura 2.1: Sistemas complexos e Sistemas Multiagentes

2.2 Funções, objetos e componentes. Agentes?

A Engenharia de Software evolui principalmente de acordo com o aumento da complexidade dos problemas que ela se propõe a resolver. Uma das ferramentas utilizadas para abordar e gerenciar esta complexidade é a Abstração [3]. O processo de abstração está relacionado à definição de um modelo simplificado do problema que enfatiza alguns detalhes ou propriedades às quais se considera mais relevantes em um dado momento [Jen99]. Os problemas tratados pela Engenharia de Software vêm se tornando cada vez mais complexos e as entidades de abstração utilizadas por esta engenharia acompanham esta evolução.

A abordagem estruturada, por exemplo, surgiu para lidar com problemas que resultavam em sistemas tipicamente centralizados, tais como os sistemas comerciais de controle de estoque tradicionais, folha de pagamento etc. Para estes sistemas, a abstração de funções e tipos de dados se mostrava adequada, uma vez que a decomposição das funcionalidades estava diretamente ligada à descrição dos requisitos do problema - em um sistema de controle de estoque, por exemplo, o requisito *Retirar produto do estoque* pode ser diretamente mapeado

[3] Além da Abstração, a Decomposição e a Organização são utilizadas como ferramentas para gerenciar a complexidade de problemas [Jen99]

para uma função. Desta forma, após a decomposição do problema em subproblemas mais específicos, já se tem as funcionalidades correspondentes a estes subproblemas, assim como o suporte de linguagem para a implementação daquela funcionalidade.

Com o advento das aplicações corporativas distribuídas de grande porte, a alta granularidade da abordagem estruturada tornou complexa a tarefa de desenvolvimento e manutenção do software. Parte disso se deve ao fato de que as funções e estruturas de dados não estão associados a uma entidade de abstração. Com a abordagem orientada a objetos, estas funções e estruturas foram associados e encapsulados em entidades do domínio do problema - os objetos. Sendo assim, criam-se dois níveis de visualização do problema: um global, onde objetos estão relacionados; e um local, onde cada uma das funções e estruturas de dados encapsulados nos objetos são implementados. Desta maneira, um problema antes decomposto em centenas de funções, por exemplo, pode ser visualizado como um conjunto de entidades que encapsulam estas funções. Reduzindo-se a granularidade da entidade de abstração, torna-se mais simples gerenciar a complexidade do problema.

Além disso, a orientação a objetos através de conceitos como Herança e Composição, apresentava-se como uma solução para atingir reutilização de software. Porém, a utilização desta abordagem ao longo dos anos não alcançou o nível de reutilização esperado pelos desenvolvedores e exigido pelo mercado [Gri94]. Foi com esta motivação que surgiu o desenvolvimento baseado em componentes(DBC) [Crn01]. Porém, em DBC não são propostas novas abstrações para a resolução do problema. Uma vez que o foco é reutilizar, definem-se diretrizes para a identificação, definição de dependências e acoplamento de funcionalidades com o objetivo de se obter um maior grau de reutilização. Apesar de tecnologias como Enterprise Java Beans(EJB) [RAJ01] e CORBA [Pag03] contemplarem o conceito de "Componente-Entidade" [4], a definição de Componente não contempla este conceito. Sendo assim, não se pode considerar um componente como uma evolução em termos de entidade de abstração e sim, como uma ferramenta para prover reusabilidade e facilitar a tarefa de manutenção do software [Szy98].

Nos dias atuais, a crescente competitividade de mercado aliada à popularização da Internet, dá origem a sistemas de larga escala com características de distribuição e heterogenei-

[4]Componentes que além de possuírem serviços e dependências também encapsulam um estado, podendo representar objetos de negócio. No caso de EJB, *EntityBeans*.

dade. Estes sistemas são inerentemente complexos [Joe02] e seus subsistemas podem estar espacialmente distribuídos, implementados em diferentes linguagens e possuírem domínios de conhecimento distintos. No desenvolvimento destes sistemas podem ser considerados dois níveis de abstração: um relacionado às funcionalidades internas de cada subsistema; e outro relacionado à interação entre os subsistemas, levando em conta as características de distribuição e heterogeneidade citadas anteriormente.

Considerando cada um dos subsistemas como entidades autônomas independentes, uma entidade de abstração adequada ao desenvolvimento deste tipo de sistema, deve contemplar a propriedade de autonomia. Desta forma, a interação entre estas entidades autônomas representaria a interação entre os subsistemas. Logo, as características implementadas pelos subsistemas, tais como a capacidade de interagir considerando heterogeneidade e domínios de conhecimento distintos, já seriam características das entidades de abstração e, portanto, poderiam ser abstraídas.

O raciocínio descrito acima é complementar ao que resultou na definição de sistemas complexos e na sua adequação à abordagem multiagentes, tal como apresentado na Seção 2.1. Na referida seção, a abordagem multiagentes foi declarada como adequada ao desenvolvimento de sistemas complexos, conforme definição e justificativas apresentadas. Agora, justifica-se a utilização da abordagem através dos atributos intrínsecos à entidade de abstração "Agente", tais como linguagens de alto nível e ontologias.

Na abstração de objetos estas características também podem ser implementadas através da utilização de ferramentas como *middlewares* [BEK+00; Gro03] e padrões de projeto [GHJV95], porém elas não são contempladas na definição de objetos. Por isso, ao abordar o problema através da abstração orientada a objetos, estas características de implementação que dizem respeito aos subsistemas teriam que ser levadas em conta, mesmo quando a preocupação é com a interação entre eles.

Concluindo, da mesma forma que a orientação a objetos encapsula funções e estruturas de dados em objetos, criando dois níveis de abstração, a abordagem multiagentes encapsula em um agente propriedades como autonomia, distribuição e heterogeneidade, criando também dois níveis de abstração:

- o nível externo, que diz respeito à interação entre as entidades de cada abordagem - objetos em Orientação a Objetos(OO) e agentes na abordagem multiagentes(MA);

- e o nível interno, que diz respeito à estrutura interna de cada uma das entidades - funções e estruturas de dados dentro das entidades de OO e objetos em MA.

Na Figura 2.2 são ilustrados estes dois níveis de abstração nestas abordagens, relacionando-as com as características das abordagens antecedentes.

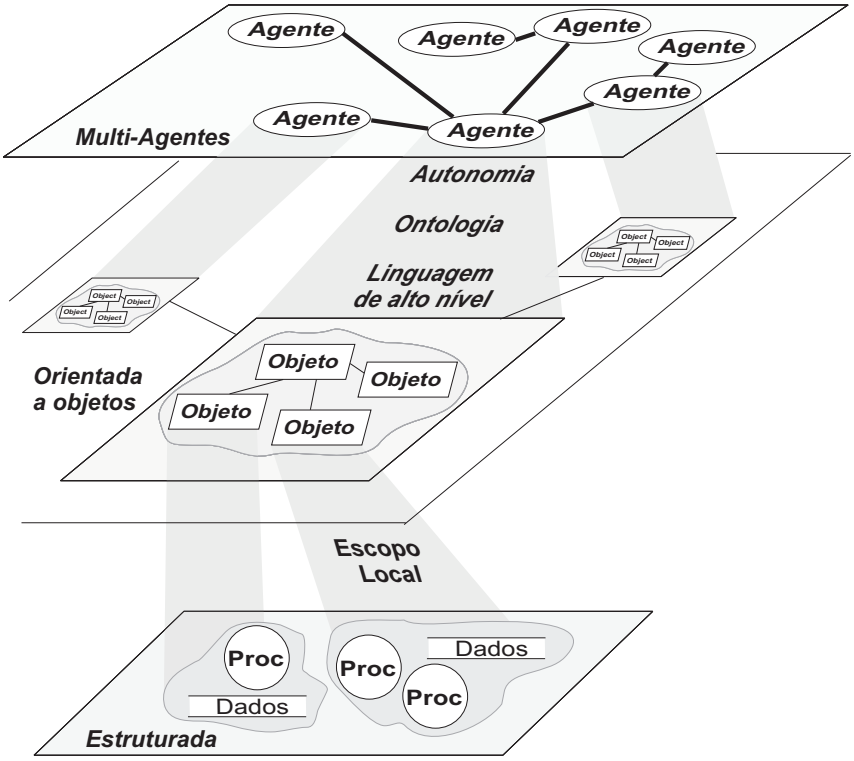

Figura 2.2: Níveis de abstração nas abordagens de Engenharia de Software

2.3 Gerência da complexidade em sistemas abertos e dinâmicos

Na Seção 2.2 a abordagem multiagentes foi declarada como adequada ao desenvolvimento de sistemas complexos devido ao fato de que sua entidade de abstração(Agente) possui atributos relacionados às entidades que compõe estes sistemas. Porém, em sistemas complexos abertos e dinâmicos, a gerência da complexidade deve estar associada a cenários de evolução, com refinamento de requisitos e conseqüentes alterações nos subsistemas e nos relacionamentos entre eles. Segundo Sussman [Jos02], esta gerência é alcançada através da flexibilização dos

relacionamentos entre as entidades que compõem o sistema. Desta forma, a capacidade do sistema de se adaptar a mudanças nos seus inter-relacionamentos é proporcional à flexibilidade de sua solução.

Nesta seção, enfatiza-se a flexibilidade da solução resultante da utilização de ferramentas como linguagens de alto nível e ontologias. Estas ferramentas, definidas como atributos da entidade Agente na Seção 2.2, fornecem à solução uma maior capacidade de adaptação diante de cenários evolutivos.

2.3.1 Interação utilizando linguagens de alto nível

Um mecanismo de interação entre dois sistemas é composto por uma linguagem e por um meio de comunicação que a transmite. Duas possíveis soluções para a definição do mecanismo de interação entre os subsistemas de um sistema complexo são descritas a seguir.

1. *Cada subsistema define o mecanismo pelo qual outros subsistemas devem interagir com ele.* Neste caso, cada subsistema define a sintaxe e a semântica de interação, as quais devem ser compreendidas pelos sistemas interessados em interagir com ele. Na medida em que cresce a quantidade de subsistemas e, conseqüentemente, a quantidade de definições de interação sintáticas e semânticas, tem-se um maior esforço para integrar os subsistemas.

2. *Todos os subsistemas utilizam uma linguagem comum, compartilhando sua sintaxe e semântica.* Desta forma, esta linguagem funciona como um padrão de sintaxe e semântica para todos os subsistemas, permitindo que o crescimento no número de subsistemas não implique na compreensão de novas definições sintáticas e semânticas por parte dos sistemas existentes.

Sendo assim, o uso de uma linguagem compartilhada para a interação entre os subsistemas provê maior flexibilidade diante de cenários evolutivos. A abordagem multiagentes utiliza este tipo de linguagem para realizar as interações entre os agentes. Exemplos de tais linguagens são FIPA-ACL [fIPA02] e KQML [FFMM94].

2.3.2 Compartilhamento de conhecimento

Quando dois (sub)sistemas interagem, eles trocam informações ou conhecimento sobre um determinado conceito. É necessário que haja um consenso em relação ao significado e representação deste conceito para evitar ambigüidades. O conceito de *Passagem*, por exemplo, possui diferentes atributos em relação a Avião, Ônibus e Navio. Uma integração entre estas empresas, com sistemas de informação e de armazenamento de dados diferentes, trará inconsistências e ambigüidades em relação ao significado do termo *Passagem*. Este tipo de distinção de significado deve ser considerado durante a interação entre os sistemas.

Utilizando o mesmo raciocínio aplicado aos mecanismos de interação, pode-se obter este consenso de duas formas, as quais são descritas a seguir.

1. *Através da difusão do conhecimento de cada subsistema aos outros.* Esta solução obriga que cada subsistema compreenda os conceitos de cada um dos outros subsistemas. Sendo assim, em cenários onde há um grande número de subsistemas, esta abordagem torna-se impraticável.

2. *Através da definição de um conhecimento compartilhado por todos os subsistemas.* Isto pode ser alcançado através de linguagens que permitam expressar conceitos, tanto sintaticamente quanto semanticamente. Uma forma de especificação de conceitos e seus relacionamentos que tem sido muito utilizada no domínio de integração de sistemas é Ontologia [Ver96].

A utilização de ontologias para a especificação dos conceitos compartilhados pelos subsistemas provê maior facilidade de integração dos subsistemas. Desta forma, novos subsistemas podem ser integrados desde que utilizem o padrão de representação e conceitualização da ontologia. O uso de ontologias também é incorporado à abordagem multiagentes, principalmente através de linguagens como DAML+OIL [DAM00], RDF [RDF97] e KIF [Gen98].

2.4 Engenharia de Software para sistemas multiagentes

Uma vez justificada a abordagem multiagentes para o desenvolvimento de sistemas de software complexos, abertos e dinâmicos, apresenta-se nesta seção uma visão geral sobre os

investimentos de engenharia para dar suporte a utilização em larga escala desta abordagem no desenvolvimento de software.

2.4.1 Rumo à Engenharia de Software Orientada a Agentes - ESOA

Os investimentos de engenharia para a concepção de software baseados em agentes tiveram início com base em duas vertentes principais, nas quais se fundamentaram os seus conceitos: a Engenharia de Software, mais especificamente, a abordagem orientada a objetos; e a engenharia do conhecimento [IGG99].

As abordagens baseadas na orientação a objetos reutilizam conceitos como herança e composição para a definição dos agentes e organizações dos sistemas. Nestas abordagens a entidade agente é definida como um objeto ativo, que encapsula a noção de estado mental [Sho93]. Exemplos de tais abordagens são as metodologias propostas por Kendall *et al.* [KMJ95], Kinny *et al.* [KGR96] e Burmeister [Bur96]. Dentre as vantagens apontadas para a extensão de conceitos de orientação a objetos estão a popularidade da abordagem OO e a possibilidade de reutilização das ferramentas e notações largamente utilizadas, tais como UML [BRJ00]. Entretanto, um objeto não é um agente e, como mostrado anteriormente, não possui propriedades necessárias para abordar sistemas com um alto nível de complexidade.

Por outro lado, as abordagens que herdam características da engenharia do conhecimento atribuem à entidade Agente um caráter mais cognitivo, provendo meios para a modelagem do seu conhecimento. Em geral, os agentes possuem mecanismos de aquisição de conhecimento e estão associados a uma ontologia. Exemplos destas abordagens são CoMoMAS [Gla97] e MAS-CommonKADS [IGG98]. Nestas abordagens, os agentes estão explicitamente relacionados ao cumprimento de tarefas. Um agente não é considerado uma abstração de software mas sim uma entidade resolvedora de problemas, com características cognitivas. Desta forma, a complexidade colocada sobre a entidade de abstração pode exceder a necessidade dos sistemas.

O ideal é unir as duas abordagens em uma outra que considere um agente como uma entidade de abstração, com características diferentes daquelas de objetos - tais como as que foram descritas na Seção 2.1, mas com complexidade definida por demanda do problema. Desta forma, utiliza-se a abstração multiagentes para desenvolver sistemas complexos, usando as ferramentas de engenharia de conhecimento como acessórios, os quais serão

utilizados de acordo com o nível de complexidade do sistema. É nesta abordagem de engenharia que se baseia a ESOA [Jen99], à qual estão vinculadas muitas das metodologias mais recentes, tais como Gaia [ZJW03], MADS [Sod01] e MaSE [DWS01].

2.4.2 Dinamicidade na ESOA? Uma infra-estrutura para a ESSMA

Zambonelli em [ZP04] descreve sua visão de complexidade vinculada à dinamicidade com base em cenários como sistemas de controle de tráfego, redes de sensores e robôs, dentre outros. Nestes cenários a adaptação do sistema às mudanças deve ser automática, ou seja, as mudanças são imprevisíveis e o sistema deve se auto-adaptar.

No trabalho apresentado nesta dissertação, considera-se que a característica dinâmica e aberta dos sistemas pode ser vista como uma conseqüência do refinamento de seus requisitos. Por exemplo, considerando os espaços de problema e solução ilustrados na Figura 2.1, pode-se mapear as entidades representativas do espaço de problema em agentes do sistema multiagentes no espaço de solução. Sendo assim, o refinamento dos requisitos sobre o espaço do problema será refletido na entrada de agentes, alteração de relacionamentos entre os agentes e das funcionalidades dos agentes que pertencem ao espaço de solução. Portanto, o refinamento de requisitos faz com que o sistema multiagentes necessite ser adaptado.

As abordagens baseadas nos conceitos da ESOA não levam em conta esta dinamicidade do sistema multiagentes [5]. Apesar de não ficar explícita na literatura a não dinamicidade da ESOA enquanto engenharia, no trabalho aqui proposto, o termo ESSMA é utilizado para designar a Engenharia Software para Sistemas Multiagentes. Considere-se ESSMA similar à ESOA, ressaltando a característica dinâmica do sistema diante de cenários evolutivos, hoje em dia, tão freqüentes.

Na Figura 2.3 é apresentado o refinamento de requisitos no espaço de problema refletido no espaço de solução. Os requisitos *a priori* definidos são refinados a cada ciclo de desenvolvimento e estes refinamentos acarretam mudanças que devem ser refletidas no espaço de solução. Esta dinâmica no espaço de solução exige que o sistema seja suficientemente flexível para que, com o mínimo de alteração, se adapte ao refinamento dos requisitos.

[5] A versão mais recente da metodologia Gaia [ZJW03], utiliza o conceito de Organizações para garantir esta dinamicidade. Porém isto é válido apenas em termos de análise. Não há suporte em relação à implementação (Ver Apêndice B).

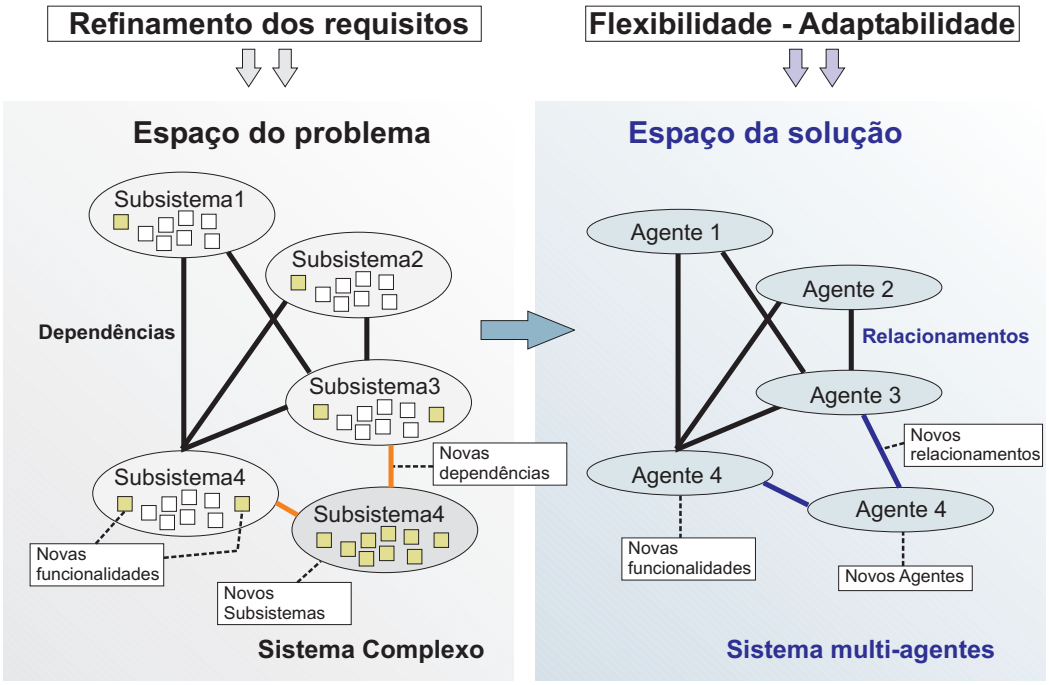

Figura 2.3: Refinamento dos requisitos, adaptação do sistema multiagentes

É com base neste princípio de dinamicidade e adaptabilidade que definem-se as seguintes características que devem ser contempladas pela ESSMA:

- *Flexibilidade para adicionar, remover e "trocar" agentes do sistema, possivelmente em tempo de execução.* Garantir a característica aberta e dinâmica do sistema multiagentes permitindo a adição, alteração e remoção dos agentes do sistema. Apesar desta também ser uma característica dependente da arquitetura do sistema, a infra-estrutura deve dar suporte para que isto seja possível.

- *Flexibilidade para alterar os relacionamentos do sistema multiagentes.* Possibilitar que agentes possam alterar seus relacionamentos sem grandes impactos sobre os outros relacionamentos. Este tipo de flexibilidade é muito dependente da arquitetura do sistema multiagentes, mas a infra-estrutura deve dar suporte para que, diante de um bom desenvolvedor, isto seja possível.

- *Flexibilidade para adicionar, alterar e remover as funcionalidades providas pelos agentes, possivelmente em tempo de execução.* Permitir que, dados os refinamentos de requisitos, funcionalidades sejam adicionadas, alteradas ou removidas dos agentes

sem grande impacto sobre as outras funcionalidades e sobre o sistema multiagentes como um todo.

Estas características foram utilizadas como requisitos-base para a infra-estrutura proposta nesta dissertação, fazendo com que a flexibilidade esteja presente em cada um dos investimentos descritos neste documento.

Capítulo 3

Diretrizes metodológicas - COMPOR-M

Além de contemplar as características da ESSMA descritas anteriormente, as diretrizes metodológicas COMPOR-M [1] têm como objetivo reduzir o espaço entre análise, projeto e a implementação do software multiagentes. Para isso, as fases clássicas de desenvolvimento de software foram contempladas e transições entre estas fases foram bem delineadas para que se tenha uma seqüência objetiva em busca da concretização do software multiagentes.

3.1 Considerações preliminares

Algumas considerações preliminares sobre os alicerces conceituais das diretrizes metodológicas são necessárias para um melhor entendimento das mesmas. Estas considerações são apresentadas a seguir.

3.1.1 Adaptando o modelo MATHEMA

Modelagem do domínio de conhecimento

A base conceitual das diretrizes aqui apresentadas, principalmente em relação à fase de análise, é o modelo MATHEMA. Este modelo é fundamentado em uma visão tridimensional do domínio de conhecimento que conduz a um particionamento deste domínio. Com base neste particionamento, define-se um sistema multiagentes. Tal como visto anteriormente, o referido modelo foi inicialmente projetado para a concepção de Sistemas Tutores Inteligentes

[1] Abreviação para COMPOR- *Methodology*

seguindo uma abordagem multiagentes.

A visão tridimensional do domínio é apresentada na Figura 3.1. Um Domínio de conhecimento, denotado por D, é particionado em sub-domínios (dij) escolhidos com base na visão de D, segundo as noções de Contexto e Profundidade. Além disso, define-se o conceito de lateralidade como um conhecimento de suporte em cada contexto, como apresentado na Figura 3.1(a).

Em [CPF98] é apresentada a modelagem do domínio de lógica clássica, D = Lógica clássica. São definidos três contextos: C1 = visão axiomática; C2 = visão de dedução natural; e C3 = visão algébrica. Possíveis profundidades e lateralidades seriam, respectivamente, d11 = Lógica clássica proposicional (ordem 0), d12 = Lógica clássica de predicado (ordem 1) e L111 = Teoria de conjuntos.

A definição de uma sociedade de agentes com base na análise do domínio, segundo uma visão tridimensional, é obtida com o mapeamento em um esquema de visões estabelecido por pares Contexto-Profundidade. Estes pares orientam a partição do domínio D em sub-domínios sobre os quais são definidos agentes, que são responsáveis pela resolução de tarefas nesta partição do conhecimento. As lateralidades também são mapeadas em agentes, como apresentado na Figura 3.1(b).

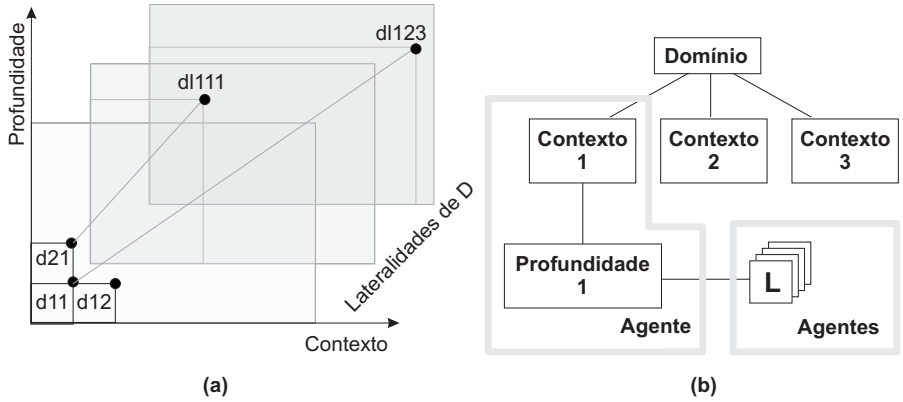

Figura 3.1: Visão tridimensional do domínio de conhecimento

Uma vez que o modelo MATHEMA é voltado para tutores inteligentes, a modelagem do domínio tem foco apenas sobre o particionamento do conhecimento. Para alcançar a generalização do modelo para SMA's em geral, o conceito de Lateralidade relativo ao conhecimento de suporte e a especialização de Contextos em Profundidades foram redefinidos em um conceito único de Habilidade, cuja definição é apresentada na Seção 3.2.1. Cada Habilidade

está vinculada a um contexto e portanto possui um conhecimento relacionado, mas também representa uma abstração para uma funcionalidade a ser provida, o que resultará em uma ou mais funcionalidades providas pelo sistema.

Arquitetura de agentes

A arquitetura dos agentes COMPOR é baseada na arquitetura de agentes proposta no modelo MATHEMA [CPF98] e é apresentada com detalhes no Capítulo 5. Na arquitetura MATHEMA, um agente é uma entidade composta por três sistemas: Sistema Tutor, Sistema Social e Sistema de Distribuição. Para a adaptação ao COMPOR, o Sistema Tutor foi renomeado para Sistema Inteligente, pois o estereótipo "Tutor" diz respeito ao domínio de tutores inteligentes. Os sistemas referentes à arquitetura de agentes COMPOR são descritos a seguir [CAPP03].

- Sistema Inteligente (SI): inclui a exibição de comportamento inteligente, representado pelas funcionalidades referentes ao domínio da aplicação e características, tais como autonomia, aprendizado, inferência de conhecimento, entre outras.
- Sistema Social (SS): inclui as funcionalidades de gerência de interação entre os agentes tais como os protocolos e linguagens de interação.
- Sistema de Distribuição (SD): inclui as funcionalidades de comunicação através de tecnologias como RMI [Mic03a] e CORBA [Gro03], por exemplo.

A utilização desta arquitetura para a concepção e implementação de agentes é apresentada no Capítulo 5.

3.1.2 Agentes baseados em componentes

A outra base conceitual para as diretrizes metodológicas, mais especificamente para as fases de projeto e implementação, é o Desenvolvimento Baseado em Componentes [Crn01]. Como descrito no Capítulo 2, uma das principais contribuições da abordagem multiagentes, em relação à Engenharia de Software, é a gerência da complexidade. Porém, a flexibilidade a partir da qual se torna viável esta gerência deve ser garantida nas fases de projeto e implementação do software. Além disso, muitas das características dos agentes, as quais foram

descritas no Capítulo 2, são comuns às diversas aplicações multiagentes. Sendo assim, o conceito de reutilização torna-se não só aplicável, como desejável.

Com a motivação destes requerimentos de flexibilidade e reusabilidade [Szy98], a abordagem baseada em componentes é utilizada para dar suporte às fases de projeto e implementação aqui descritas, mais especificamente, na composição das funcionalidades dos agentes COMPOR.

3.2 Fases de desenvolvimento

As diretrizes metodológicas COMPOR-M para o desenvolvimento de sistemas multiagentes contemplam quatro fases clássicas do desenvolvimento de software: análise, projeto, implementação e testes. O desenvolvimento é iterativo/incremental e apesar da semelhança da Figura 3.2 com a representação gráfica do modelo cascata [Pre95], não está vinculado a nenhum modelo de desenvolvimento. As diretrizes determinam apenas as atividades de cada fase, com transições seguindo a seqüência clássica de desenvolvimento de software. As fases contempladas pelas diretrizes são apresentadas na Figura 3.2 e são descritas a seguir.

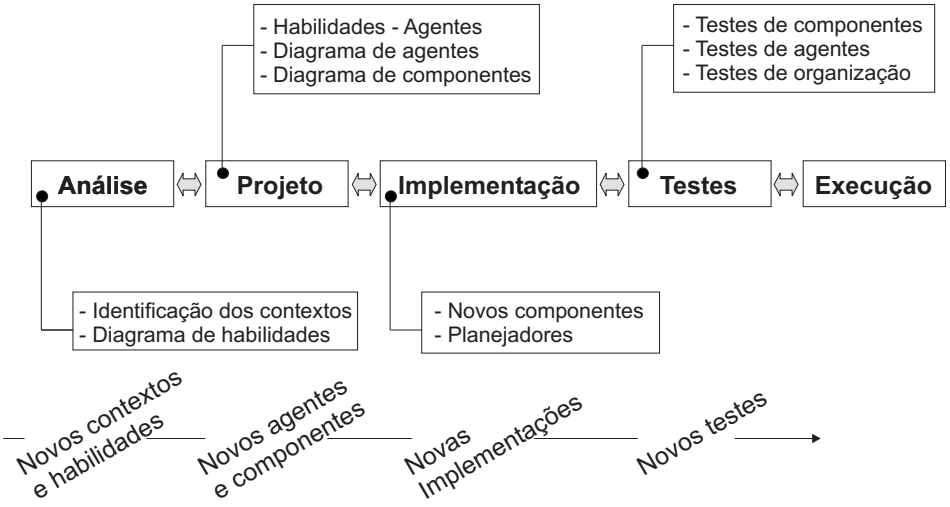

Figura 3.2: fases contempladas pelas diretrizes COMPOR-M

3.2.1 Fase de análise

As diretrizes de análise são baseadas na análise de domínio MATHEMA. Suas diretrizes foram generalizadas para englobar os diversos domínios de aplicação de sistemas multiagentes. Algumas definições se fazem necessárias para o entendimento das diretrizes de análise. Estas definições são apresentadas a seguir.

Definição 3.1 (Domínio) *Um Domínio (D) do problema representa a área de conhecimento na qual um determinado problema se enquadra.*

Definição 3.2 (Contexto) *Um Contexto (C_i) representa uma visão específica do Domínio (D), a partir da qual define-se uma partição especializada deste domínio, associada a uma ontologia que descreve o conhecimento e o vocabulário relativo a esta partição.*

Definição 3.3 (Habilidade) *Uma Habilidade (H_{ij}) representa um conjunto não vazio de funcionalidades necessárias para a resolução de um determinado (sub)problema referente ao contexto "i" específico* [2].

Uma vez definidos Domínio, Contexto e Habilidade, as diretrizes de análise são descritas a seguir.

1. Dado um domínio de problema (D), definem-se os Contextos (C_i) inerentes a ele, os quais representam um particionamento do domínio.

2. Para cada Contexto (C_i) define-se uma ontologia que representa o conhecimento e vocabulário associado. Esta ontologia pode ser uma partição da ontologia do domínio.

3. Para cada Contexto (C_i), identificam-se as Habilidades (H_{ij}) necessárias para a resolução do problema.

4. Definir as dependências existentes entre as habilidades.

As habilidades, suas dependências e os contextos a que pertencem são descritas através de um Diagrama de Habilidades como apresentado na Figura 3.3.

[2] Este conceito é semelhante ao conceito de casos de uso na análise orientada a objetos [Lar00]. Porém, uma Habilidade está inserida em um escopo específico - o contexto

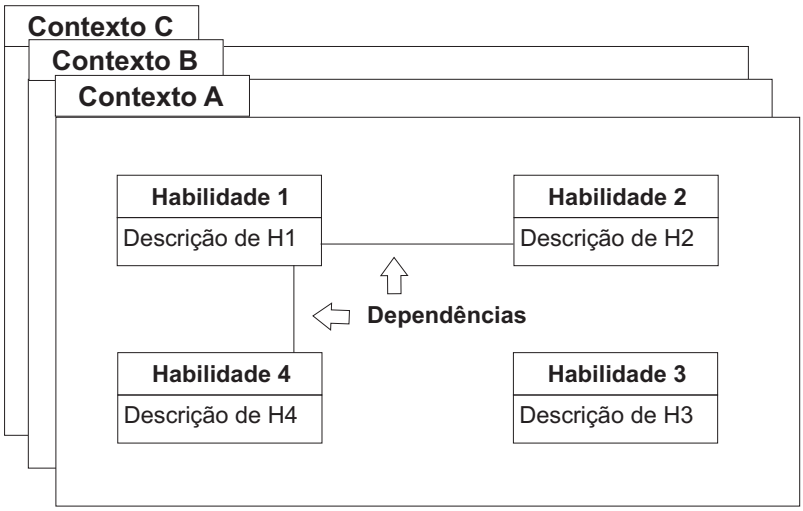

Figura 3.3: Diagrama de Habilidades

3.2.2 Fase de projeto

Uma vez identificadas as habilidades necessárias em cada contexto para a resolução do problema, pode-se então definir o projeto arquitetural da sociedade de agentes, assim como a arquitetura interna de cada agente. As diretrizes de projeto são apresentadas a seguir.

1. Mapear cada Contexto (C_i) definido em análise para uma Organização [3] (O_i). A ontologia definida para este contexto passa a ser vista como um conhecimento disponível e compartilhado pelos integrantes da organização.

2. Alocar as habilidades, definidas anteriormente para cada contexto, a Agentes (A_{ij}) pertencentes às organizações, ou seja, definir que agentes contemplam quais habilidades. O mapeamento é do tipo *n para m*.

3. Definir o(s) objetivo(s) de cada agente em relação às organizações a que pertencem e as habilidades que contemplam.

4. Mapear as dependências entre as habilidades em associações entre os agentes que as contemplam. Cada associação representa uma ou mais interações entre os agentes, de acordo com os protocolos definidos em 5.

[3] No contexto deste trabalho uma organização é vista como um conjunto de agentes em interação, como descrito no Capítulo 2

5. Definir a arquitetura do sistema e os protocolos de interação entre os agentes de acordo esta arquitetura [4]. De acordo com os papéis definidos nos protocolos de interação, atribuir estes papéis aos agentes.

O resultado da aplicação das diretrizes de projeto de 1 a 5 são representados em um Diagrama de Agentes, como ilustrado na Figura 3.4.

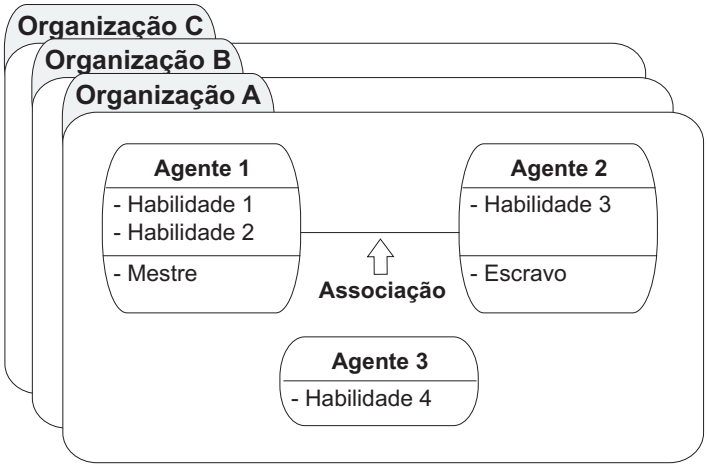

Figura 3.4: Diagrama de Agentes

Uma vez definidos os agentes que compõem a arquitetura das organizações, pode-se então definir a arquitetura interna dos agentes, identificando seus componentes funcionais. Esta fase é similar à fase de identificação dos componentes com base em requisitos no Desenvolvimento Baseado em Componentes [Crn01]. A utilização da abordagem baseada em componentes para compor os agentes tem como objetivo maximizar a reutilização das suas funcionalidades.

6. Mapear as habilidades e os papéis de interação de cada Agente em serviços providos por componentes funcionais. Caso os componentes deste mapeamento já existam, serão reutilizados. Caso não existam, serão implementados e estarão disponíveis para reutilização em futuros sistemas [5]. As habilidades não mapeadas para componentes terão suas funcionalidades implementadas pelo componente Planejador do agente, descrito na Seção 3.2.3.

[4] Em [ON00] é apresentada uma análise de protocolos de interação entre agentes e em [Jun03] são apresentados diversos padrões arquiteturais para agentes, os quais contemplam estes protocolos.

[5] A reutilização de componentes a partir de um repositório é uma das grandes motivações do desenvolvimento baseado em componentes

7. Definir os componentes que fazem parte de cada Agente, contextualizando suas funcionalidades de acordo com o modelo de agentes COMPOR - inteligente, social ou distribuição.

Os resultados da aplicação das diretrizes 6 e 7 são representados em um Diagrama de Componentes. Neste diagrama são listados os componentes que fazem parte de cada um dos sistemas dos agentes (inteligente, social e distribuição), como apresentado na Figura 3.5.

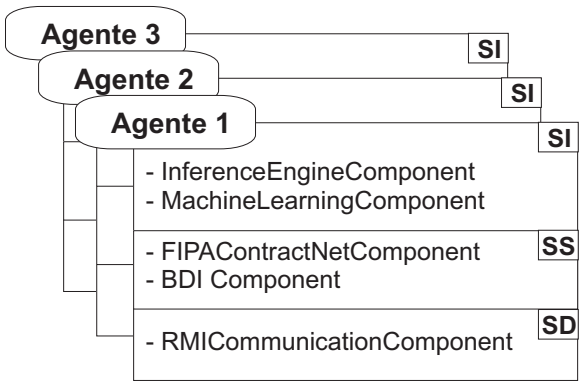

Figura 3.5: Diagrama de Componentes

3.2.3 Fase de implementação

Assim como acontece no Desenvolvimento Baseado em Componentes [Crn01], os componentes identificados na fase anterior, caso não existam, devem ser implementados. Este é o primeiro passo da fase de implementação aqui descrita. A implementação dos novos componentes deve ser feita com base em algum modelo de componentes, que ditará a forma como os componentes são construídos e depois conectados entre si. Por isso, no Capítulo 4 é apresentado um modelo de componentes concebido para dar suporte à componentização de acordo com a arquitetura de agentes COMPOR. Outros modelos existentes podem ser utilizados, mas a adaptação à arquitetura dos agentes tende a ser mais complicada.

Após a implementação de todos os componentes definidos na fase de projeto, o componente Planejador de cada agente é definido. O Planejador é um *script* de execução de cada agente que coordenará as suas funcionalidades e suas ações em relação à sociedade. No Capítulo 5 é apresentado um arcabouço para a construção de sistemas multiagentes de acordo

com o modelo de componentes do Capítulo 4. Neste arcabouço, a definição do Planejador é feita através da implementação de uma classe Java.

3.2.4 Fase de testes

O investimento na fase de testes vem se tornando indispensável no desenvolvimento de software. Como visto anteriormente, as funcionalidades dos agentes são providas pelos componentes e, sendo assim, testes em cada um dos componentes tornam-se independentes dos testes dos agentes e da sociedade como um todo. Portanto, as diretrizes para testes descritas a seguir têm como foco o teste dos agentes e do seu comportamento enquanto integrante da sociedade.

1. **Testar os serviços de cada componente funcional.** O teste dos componentes é independente da sua utilização como parte dos agentes. Sendo assim, os componentes, quando construídos, devem ser testados, tornando esta diretriz redundante.

2. **Testar as dependências entre os componentes de um determinado agente.** Os serviços providos pelos componentes funcionais devem ter sua execução testada em relação aos demais. Para tanto, as dependências de cada serviço devem ser testadas.

3. **Testar a interação entre os agentes da sociedade.** Os planejadores dos agentes devem ser testados para que o funcionamento da sociedade, mediante determinadas condições (ou parâmetros de entrada [Pre95]), seja validado.

Apesar da utilização de testes como parte do desenvolvimento de software ser antiga, o que realmente impulsionou sua utilização foram as ferramentas para a automação destes testes [BG98]. Por isso, o ambiente de desenvolvimento apresentado no Capítulo 6 possui uma ferramenta de automação de testes para facilitar a aplicação das diretrizes.

3.3 Trabalhos correlatos

Várias metodologias foram propostas para sistematizar o desenvolvimento de sistemas multiagentes. Como descrito no Capítulo 2, algumas destas metodologias são baseadas em conceitos da abordagem orientada a objetos com adaptações para a Engenharia de Software orien-

tada a agentes. Exemplos destas metodologias foram propostas por Kendall *et al.* [KMJ95], Kinny *et al.* [KGR96] e Burmeister [Bur96]. Outras metodologias se baseiam na engenharia do conhecimento tais como CoMoMAS [Gla97] e MAS-CommonKADS [IGG98]. As metodologias mais recentes, são definidas sobre o conceitos da Engenharia de Software Orientada a Agentes - ESOA [Jen99]. Exemplos destas metodologias são Gaia [ZJW03], MaSE [DWS01] e MADS [Sod01].

Em geral as metodologias não abordam todas as fases do desenvolvimento de software, tendo como conseqüência um enorme espaço entre o resultado da análise e projeto e a implementação e execução do software. A dinamicidade do sistema multiagentes não é levada em conta nas diretrizes, com exceção de Gaia [ZJW03], não prevendo mudança de papéis ou relacionamentos. Já que a dinamicidade não é levada em conta, seu impacto sobre a implementação do sistema não é analisado e, sendo assim, a fase de implementação não recebe tanta importância nestas metodologias. Apesar de considerar a dinamicidade, a metodologia Gaia não leva em conta as características de implementação

A flexibilidade do software para sistemas multiagentes é o principal objetivo das diretrizes COMPOR-M. O conceito de habilidade, por exemplo, só tem sentido de análise se houver uma infra-estrutura de software suficientemente flexível para que mudanças nas habilidades sejam refletidas em implementação, sem grandes esforços. Tanto este conceito, como outros conceitos utilizados nas diretrizes foram concebidos objetivando a flexibilidade, a qual é garantida, em implementação, pelo arcabouço de software apresentado no Capítulo 5, sendo assim, desde a definição das habilidades em análise já se objetiva uma implementação flexível.

No Apêndice B apresenta-se uma análise comparativa entre metodologias existentes e as diretrizes COMPOR-M, apontando as principais motivações para a definição de diretrizes para o desenvolvimento de software para sistemas multiagentes.

Capítulo 4

Modelo de componentes - COMPOR-CM

No Capítulo 3 foram apresentadas diretrizes metodológicas para o desenvolvimento de sistemas multiagentes. De acordo com estas diretrizes e com as definições de Agente e Sistemas Multiagentes apresentadas no Capítulo 2, as habilidades dos agentes que fazem parte de uma organização e os relacionamentos entre estes agentes têm um caráter dinâmico. Sendo assim, uma vez que, como visto na Seção 3.1.2, as habilidades dos agentes são mapeadas para serviços de componentes, torna-se necessário um modelo de componentes que viabilize tal dinamicidade - o modelo COMPOR-CM [1] [CAPP03].

Embora o modelo COMPOR-CM tenha sido criado para dar suporte à arquitetura de agentes COMPOR, também é aplicável a outros tipos de sistemas. O modelo tem base no princípio de que a inexistência de referências explícitas, ou diretas, entre provedores de funcionalidades tem como conseqüência uma maior flexibilidade na inserção, remoção e alteração destes provedores, inclusive em tempo de execução. Sistemas que possuem esta flexibilidade como requisito podem utilizar o modelo COMPOR-CM, ainda que não se utilizem da abordagem de agentes.

Dois tipos de entidade são definidas em COMPOR-CM: os **contêineres** e os **componentes funcionais**. Os componentes funcionais não possuem e implementam as funcionalidades do sistema, disponibilizando-as em forma de serviços. Os componentes funcionais não são compostos por outros componentes, ou seja, não possuem “componentes-filhos”. Os contêineres, por sua vez, não implementam funcionalidades, apenas gerenciam o acesso aos serviços dos seus “componentes-filhos”. Em resumo, os contêineres servem como “portas

[1] Abreviação para COMPOR- *Component Model*

de acesso" às funcionalidades dos componentes neles contidos.

Na Figura 4.1 apresenta-se uma arquitetura de sistema baseada em composição de módulos, onde cada módulo é composto de sub-módulos que implementam as funcionalidades do sistema, mapeada em uma estrutura onde os nós-folhas representam as funcionalidades(componentes funcionais) e os outros nós (contêineres) representam "portas de acesso" às funcionalidades.

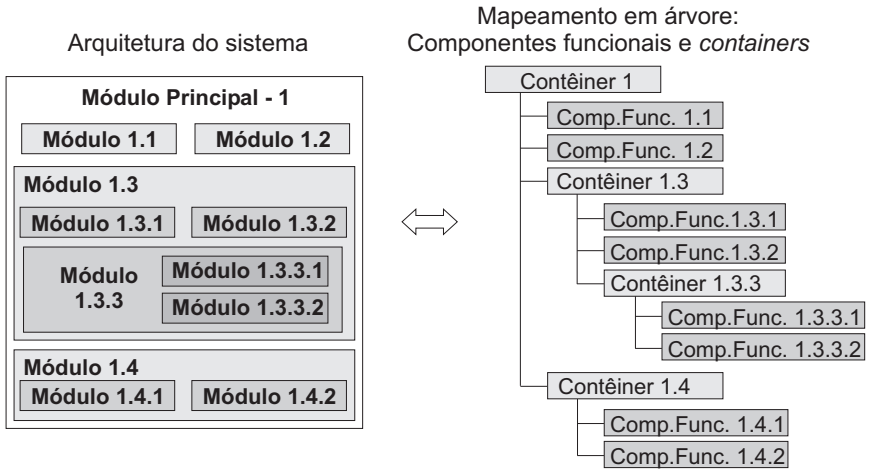

Figura 4.1: Mapeamento em árvore da arquitetura hierárquica de um sistema

Com base neste mapeamento de arquitetura, COMPOR-CM torna possível a reutilização tanto no nível de componentes funcionais como no nível de contêineres. Na Figura 4.2, por exemplo, apresenta-se a arquitetura clássica de software em três camadas na visão do modelo COMPOR-CM. Nesta figura, cada uma das camadas foram mapeadas em módulos e suas funcionalidades em sub-módulos. O sistema (visto como um contêiner) é representado por três módulos (ou três contêineres), que por sua vez possuem componentes funcionais, os quais provêem os serviços do sistema. Visualizando um contêiner como uma caixa preta, a reutilização pode ocorrer no nível de sistema, para interoperabilidade com outros sistemas, por exemplo; no nível de módulos, caso ocorram modificações no tipo de interface do usuário ou estrutura de armazenamento, por exemplo; e no nível de componentes, caso alguma funcionalidade do sistema possa ser reutilizada em outros contextos.

Em um modelo de componentes são definidos os processos de montagem ou composição, de interação e de disponibilização e especificação de componentes [Crn01]. A seguir são descritos, respectivamente, o modelo de especificação, o processo de disponibilização

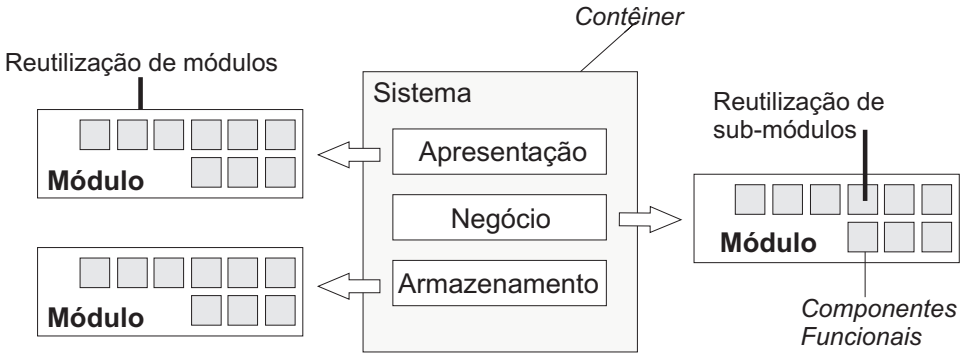

Figura 4.2: Arquitetura clássica em três camadas mapeadas em módulos

de componentes e o modelo de interação entre componentes COMPOR-CM. Depois disto, modelos de componentes correlatos a COMPOR-CM são apresentados.

4.1 Modelo de Especificação

Para viabilizar o desenvolvimento de sistemas baseado em componentes é necessária uma especificação precisa das funcionalidades de cada componente. Para que os desenvolvedores possam utilizar o componente como uma "caixa-preta" é necessário fornecer alguma descrição dos serviços que este provê. O modelo de especificação de componentes COMPOR-CM é apresentado a seguir.

1. **Nome:** o nome do componente.

2. **Descrição:** uma breve descrição dos tipos de serviços que este componente provê.

3. **Contexto:** o contexto no qual se insere o componente em relação ao sistema. No exemplo apresentado na Figura 4.2 os possíveis contextos seriam *Apresentação*, *Negócio* e *Armazenamento*. Em relação a aplicações multiagentes, na visão da arquitetura COMPOR, os contextos são *Inteligente*, *Social* ou *Distribuição*, representando os sistemas que formam a arquitetura dos agentes, como apresentado na Seção 3.1.1. Caso não haja um contexto definido para o componente, pode-se usar o contexto *Outro*.

4. **Dicionário de tipos:** definição de um dicionário de tipos abstratos a serem utilizados na especificação do componente.

5. **Atributos de inicialização:** Especificação dos atributos necessários à inicialização do componente.

6. **Serviços providos:** listar os serviços do componente, especificando os parâmetros necessários à sua execução, o resultado esperado da invocação e os possíveis erros ou exceções ocorridos durante a execução do serviço. A especificação de tipos é feita de acordo com o dicionário definido em 4.

7. **Dependências de serviços:** especificação das dependências deste componente em relação aos serviços providos por outros componentes.

8. **Eventos anunciados:** listar os eventos anunciados por este componente.

9. **Eventos de interesse:** listar os eventos de interesse do componente.

Caso seja necessário um maior detalhamento na especificação dos serviços ou eventos de um determinado componente podem ser utilizados modelos orientados a objetos, por exemplo. Este detalhamento não faz parte da especificação do componente COMPOR-CM porque a invocação de um serviço de um componente não é feita diretamente, como descrito na Seção 4.4. Sendo assim, uma vez que a gerência de invocação de serviços é feita pelos contêineres, a interface do componente funcional não é acessada pelo usuário. Isto é evidenciado na implementação do arcabouço de software que implementa a especificação do modelo COMPOR-CM, apresentado no Capítulo 5.

4.2 Disponibilização de componentes

Os componentes funcionais são disponibilizados através da inserção nos contêineres. É necessário que haja ao menos um contêiner (contêiner-raiz) para que seja possível adicionar os componentes funcionais. Cada contêiner possui uma lista dos serviços providos e eventos de interesse de cada um dos seus "componentes-filhos". Após a inserção de um componente, a lista de serviços e eventos de cada contêiner até a raiz da hierarquia é atualizada. Isto é necessário para que os serviços providos pelo componente recém-adicionado estejam disponíveis para qualquer outro componente funcional do sistema, assim como para permitir que ele seja notificado caso algum evento de seu interesse seja disparado por outro componente.

O processo de disponibilização de componentes é apresentado na Figura 4.3 e seus passos são descritos a seguir. As operações de remoção e alteração de componentes são similares.

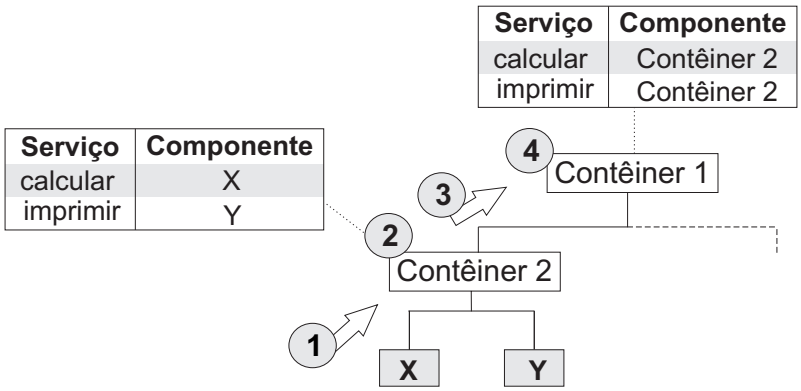

Figura 4.3: Disponibilização de componentes: atualização da lista de serviços e eventos dos "componentes-filhos" até a raiz da hierarquia

1. O componente "X", que implementa o serviço "calcular", é adicionado ao Contêiner 2.

2. O Contêiner 2 atualiza a sua tabela de serviços providos pelos seus "componentes-filhos" (o mesmo acontece em relação à tabela de eventos de interesse).

3. O Contêiner 2 solicita que o seu "contêiner-pai" (Contêiner 1) atualize a sua tabela de serviços.

4. O Contêiner 1 atualiza a sua tabela de serviços providos pelos seus "componentes-filhos".

Após este processo, os serviços do componente "X" podem ser acessados a partir de qualquer componente da hierarquia, sem referenciar "X" diretamente. A única entidade do modelo à qual um componente funcional possui referência é o seu "contêiner-pai".

4.3 Modelo de interação

Em COMPOR-CM são definidos dois tipos de interação entre componentes: baseada em serviços e baseada em eventos. Na interação baseada em eventos o foco está sobre o anúncio

da mudança de estado de um determinado componente, localizado em um determinado contêiner, aos componentes interessados, mesmo que estes estejam localizados em contêineres diferentes. A interação baseada em serviços permite a invocação dos serviços de um determinado componente a partir de qualquer outro componente do sistema, ainda que pertençam a contêineres diferentes. Em ambos os tipos de interação não há referência explícita entre os componentes, como mostrado a seguir.

4.3.1 Interação baseada em serviços

Como visto anteriormente, após a inserção de um componente em um determinado contêiner, seus serviços tornam-se disponíveis a qualquer outro componente do sistema. Sendo assim, supondo a existência do serviço “salvar” implementado pelo componente K, pode-se solicitar a execução deste serviço a partir de um componente X, sem fazer referência a K. Este processo é apresentado na Figura 4.4 e seus passos são descritos a seguir.

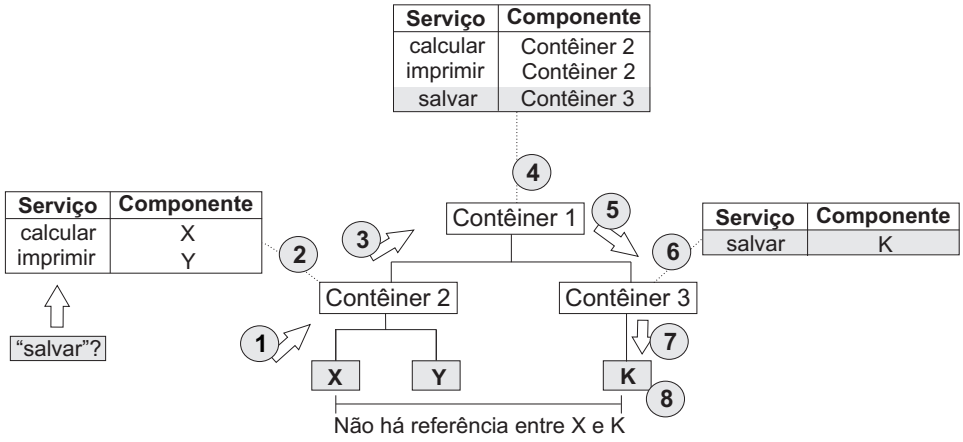

Figura 4.4: Interação baseada em serviços: localização e execução sem referência entre componentes funcionais

1. O componente “X” solicita a execução do serviço “salvar” ao seu “contêiner-pai”.
2. O Contêiner 2 verifica, de acordo com sua tabela de serviços, que nenhum dos seus “componentes-filhos” implementa o serviço “salvar”.
3. O Contêiner 2 então encaminha a solicitação ao seu “contêiner-pai” (Contêiner 1).

4. O Contêiner 1 verifica, de acordo com sua tabela de serviços, que um de seus "componentes-filhos" implementa o serviço "salvar"(Contêiner 3). Para o Contêiner 1, o Contêiner 3 é visto como um componente que implementa o serviço.

5. O Contêiner 1 então encaminha a solicitação de serviço para o Contêiner 3.

6. O Contêiner 3 não implementa o serviço mas possui em sua tabela uma referência ao real implementador do serviço - componente "K".

7. O Contêiner 3 então encaminha a solicitação de serviço para o componente funcional "K".

8. O componente "K" executa o serviço "salvar" e retorna, caso exista, o resultado da execução, o qual segue o caminho inverso da solicitação.

Como ilustrado na Figura 4.4, não há referência alguma entre o componente solicitante do serviço ("X") e o componente provedor do mesmo ("K"). Desta forma, é possível alterar o componente que provê o serviço "salvar" sem modificar o restante da estrutura.

4.3.2 Interação baseada em eventos

Quando um evento é disparado por um determinado componente funcional, toda a hierarquia de componentes do sistema deve ser verificada para que todos os interessados no evento sejam avisados. A interação baseada em eventos também é realizada pelos contêineres, não havendo referência direta entre os componentes funcionais. Na Figura 4.5 este processo é ilustrado e seus passos são descritos a seguir.

1. O componente "X" anuncia um evento denominado "EventoA".

2. O anúncio é recebido, diretamente, apenas pelo seu contêiner-pai (Contêiner 2) que verifica em sua tabela de eventos de interesse dos "componentes-filhos" se algum deles está interessado no evento.

3. O Contêiner 2 encaminha o evento ao único interessado no evento de acordo com a sua tabela - o componente "Y".

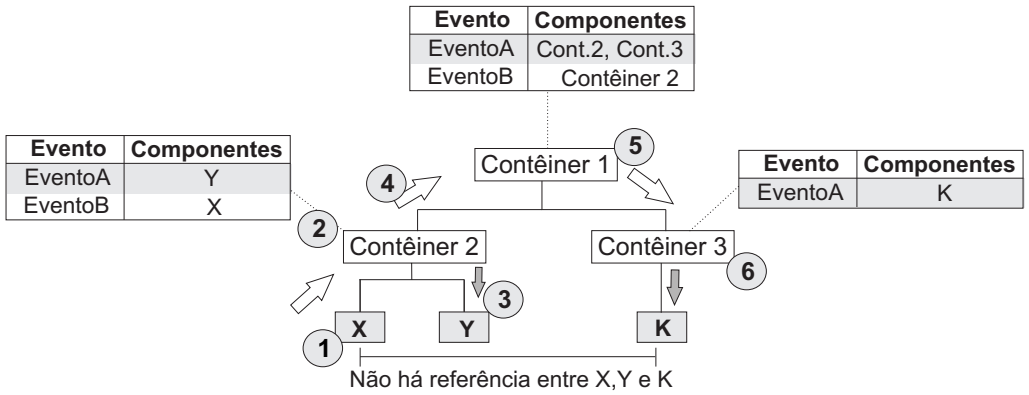

Figura 4.5: Interação baseada em eventos: notificação de eventos sem referência entre componentes funcionais

4. Além disso, o Contêiner 2 também encaminha o evento ao seu Contêiner-pai (Contêiner 1).

5. O Contêiner 1, de acordo com sua tabela de eventos, repassa o evento aos interessados, exceto para o originador do anúncio do evento (Contêiner 2). Sendo assim, encaminha o evento ao Contêiner 3. Por representar a raiz da hierarquia, ele não possui um contêiner-pai para o qual também repassaria o evento.

6. O Contêiner 3 repassa o evento ao componente interessado de acordo com a sua tabela de eventos. No exemplo, o componente “K” é o único interessado.

Como ilustrado na Figura 4.5, não há referência alguma entre o componente anunciador do evento (“X”) e os interessados no evento (“Y” e “K”). Desta forma, o componente que anuncia o evento poderia ser alterado sem modificar o restante da estrutura.

4.4 Modelos de componentes correlatos

Existem várias propostas para o desenvolvimento baseado em componentes, tais como os modelos *JavaBeans* [Mic97], *Enterprise Java Beans* [RAJ01] e CORBA Component Model [Pag03].

Tanto *JavaBeans* como *Enterprise Java Beans* utilizam referências explícitas para a interação entre componentes. Isto dificulta a gerência de atualizações em tempo de execução, já que existe referência aos componentes e não aos serviços providos por eles.

No caso de EJB, há um repositório único de componentes do sistema e a interação entre eles é definida através das definições de interface para tais componentes. Em relação à modularização, EJB possui apenas um nível de repositório para componentes e não dá suporte a “módulos compostos de módulos”. Desta forma, a reutilização “caixa-preta” se restringe ao nível de componentes de baixa granularidade. No caso da arquitetura clássica em três camadas por exemplo, pode-se reutilizar tanto um componente da camada de armazenamento como toda a camada de armazenamento, caso necessário.

No modelo de componentes CORBA (CCM), a mudança de componentes é dificultada pois as interfaces remotas estão explicitamente relacionadas. Uma vez que a especificação dos provedores de serviços é feita através das interfaces (IDL) [2], alterações nos provedores devem ser refletidas nas interfaces remotas.

De um modo geral, as tecnologias existentes possuem como foco prover suporte a características não funcionais como persistência, distribuição e segurança. Tal suporte é indispensável ao desenvolvimento de certas aplicações tais como as corporativas. Apesar destas características não serem contempladas na versão atual de COMPOR-CM, estas podem ser implementadas e acopladas ao modelo em versões futuras.

O modelo COMPOR-CM tem como foco a independência entre os componentes funcionais garantindo maior flexibilidade para mudanças, inclusive em tempo de execução, das funcionalidades de um sistema, através da inserção, remoção ou alteração de componentes. Além disso, tem-se suporte à definição de arquiteturas hierárquicas com reutilização em vários níveis, “da raiz às folhas” da hierarquia de módulos de um sistema.

[2] Linguagem de definição de interfaces (*Interface Definition Language*) utilizada para a especificação de serviços de componentes segundo o modelo de componentes CORBA [Pag03]

Capítulo 5

Arcabouço baseado em componentes - COMPOR-F

No modelo COMPOR-CM apresentado no Capítulo 4 são definidas as regras de interação entre os componentes e as convenções de especificação que devem ser seguidas para o acoplamento de novos componentes ao modelo. Como descrito em [FJS00], para implementar estas regras de interação e especificação de forma transparente ao desenvolvedor, um arcabouço de software pode ser definido. Sendo assim, neste capítulo é apresentado o arcabouço COMPOR-F [1] [APPC03] que implementa as especificações do modelo de componentes COMPOR-CM.

O arcabouço COMPOR-F tem seu projeto baseado no padrão de projeto *Composite* [GHJV95]. Este padrão de projeto é aplicável a arquiteturas hierárquicas nas quais as entidades "compostas" contêm tanto outras entidades "compostas" quanto "entidades-folha". Através da utilização de uma interface única de acesso a estas entidades, garante-se uma composição recursiva.

Na Figura 5.1 é apresentado o diagrama de classes do arcabouço COMPOR-F utilizando a linguagem UML [BRJ00]. A composição recursiva apresentada neste modelo se baseia no conceito de interface, o qual é definido em linguagens de programação como Java, e foi descrita em [Gra99]. Este mesmo modelo pode ser facilmente implementado em outras linguagens como C++, utilizando o conceito de classes abstratas e herança múltipla, como apresentado em [GHJV95].

[1] Abreviação para COMPOR- *Framework*

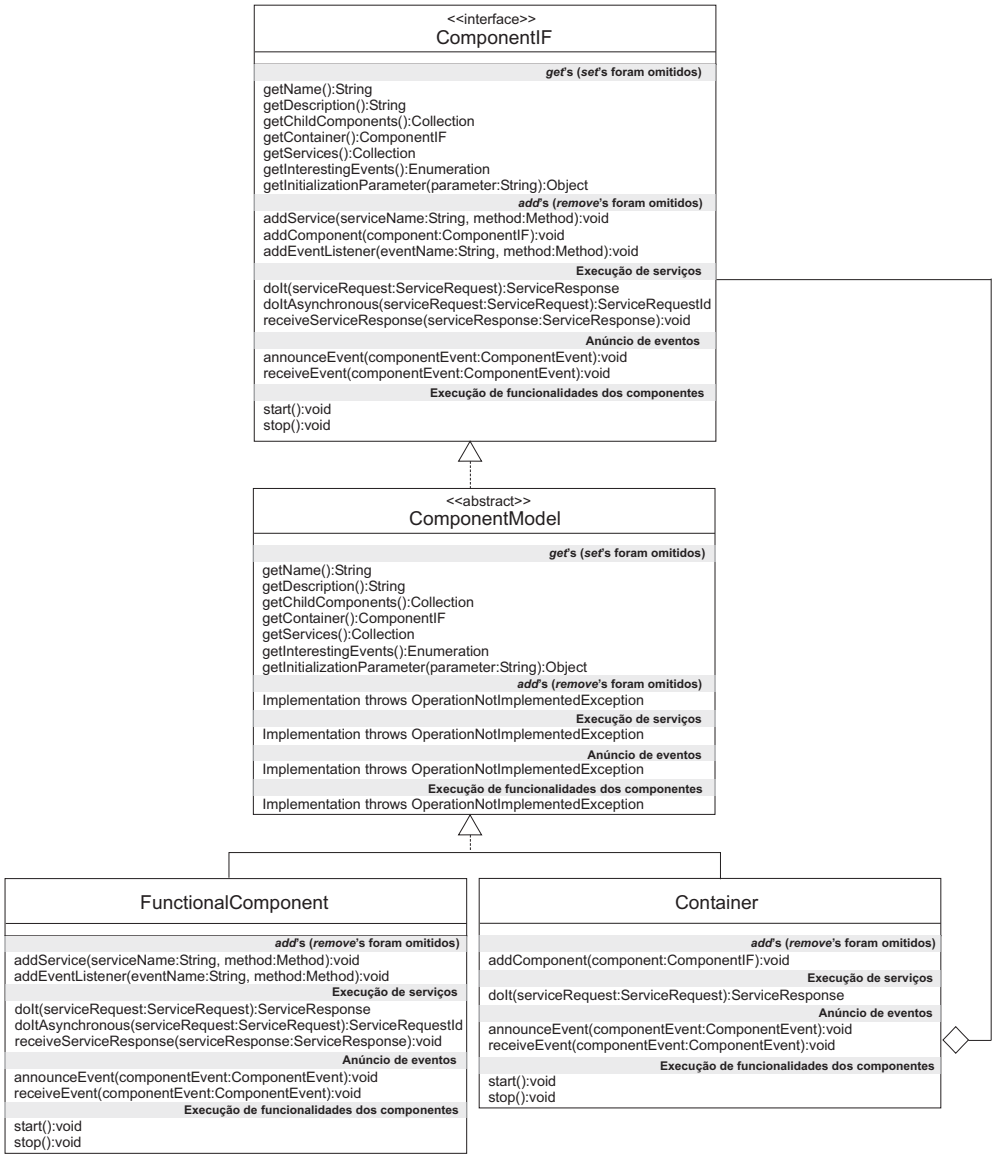

Figura 5.1: Diagrama de classes do arcabouço COMPOR-F

As classes *FunctionalComponent* e *Container* são instanciadas, respectivamente, em componentes funcionais e contêineres, os quais são conceitualizados no modelo COMPOR-CM.

A interface *ComponentIF* garante a composição recursiva, permitindo que os contêineres "desconheçam" a implementação dos seus "componentes-filhos", que podem ser tanto componentes funcionais como outros contêineres.

A classe abstrata *ComponentModel* implementa os métodos comuns às suas subclasses, como os métodos de acesso a atributos (*get* e *set*). Além disso, define pseudo-implementações dos seus métodos, com base no padrão de projeto *Adapter* [GHJV95], para

que as classes *Container* e *FunctionalComponent* implementem apenas os métodos pelos quais são responsáveis, de acordo com o modelo de componentes COMPOR-CM.

Os demais métodos declarados na interface *ComponentIF* têm diferentes implementações para *FunctionalComponent* e *Container*. Estas implementações, apresentadas a seguir, refletem a especificação de disponibilização de componentes, interação baseada em serviços e baseada em eventos do modelo COMPOR-CM descritas no Capítulo 4.

5.1 Implementação da disponibilização de componentes

O processo de disponibilização de componentes descrito na Seção 4.2 é implementado pelos métodos *addComponent*, *addService* e *addEventListener*. A seguir é apresentada uma descrição da funcionalidade destes métodos assim como suas respectivas implementações para as classes *Container* e *FunctionalComponent*.

- ***addService***: adiciona um novo serviço a um componente, definindo um método implementador deste serviço. O método definido será invocado para a execução do serviço.

  Container: este método não é implementado por contêineres.
  FunctionalComponent: armazena o método definido como implementador de serviço em sua tabela de serviços [2].

- ***addEventListener***: adiciona um interessado em um determinado evento de um componente, definindo um método receptor do evento. O método definido será invocado quando o evento for disparado por outro componente do sistema.

  Container: este método não é implementado por contêineres.
  FunctionalComponent: armazena o método definido como receptor do evento em sua tabela de eventos de interesse.

- ***addComponent***: adiciona um novo componente a um componente.

  Container: adiciona os eventos e serviços do componente às suas tabelas de eventos de interesse e de serviços providos pelos seus "componentes-filhos".
  FunctionalComponent: este método não é implementado por componentes funcionais.

[2] Na linguagem Java, o armazenamento e invocação de métodos é realizado através da classe *Method* da API(*Application Programming Interface*) *Reflection* [Mic03b]

5.2 Implementação do modelo de interação baseada em serviços

O modelo de interação baseada em serviços descrito na Seção 4.3.1 é implementado em duas versões no arcabouço COMPOR-F: síncrona e assíncrona.

A implementação da interação síncrona, que reflete exatamente o modelo definido na Seção 4.3.1, é realizada através de invocações sucessivas do método *doIt* na hierarquia de componentes até que o componente funcional provedor do serviço seja localizado. A seguir é apresentada uma descrição da funcionalidade deste método assim como suas respectivas implementações para as classes *Container* e *FunctionalComponent*.

- ***doIt***: executa um serviço provido por algum componente do sistema. Este método é invocado a partir de qualquer componente do sistema e o provedor do serviço não precisa ser especificado. Os componentes podem pertencer a contêineres diferentes.

  Container: se o serviço requisitado for implementado por algum *ComponentIF* nele contido, então a execução é encaminhada através da invocação do método *doIt* neste *ComponentIF*. Caso contrário, encaminha-se a execução do serviço ao contêiner ao qual este contêiner pertence.
  FunctionalComponent: se o serviço requisitado for implementado por este componente, então executa-o. Caso contrário encaminha a execução do serviço ao contêiner ao qual este componente pertence.

A interação assíncrona entre componentes tem sua implementação baseada no padrão de concorrência *Future* [LS95]. Um componente invoca um serviço assincronamente através do método *doItAsynchronous* e recebe como retorno um identificador da requisição (*ServiceRequestId*). Então, uma nova linha de execução é iniciada pelo componente invocador com a função de solicitar o serviço via método *doIt* e aguardar o seu retorno. Quando o retorno de *doIt* chega à linha de execução recém-criada, esta invoca o método *receiveServiceResponse* no componente invocador do serviço, repassando a resposta do serviço e o identificador da requisição. De posse do identificador da requisição, o componente invocador identifica a que invocação se refere a resposta do serviço.

- ***doItAssynchronous***: este método é uma versão assíncrona do método *doIt*.

Container: este método não é implementado por contêineres.

FunctionalComponent: cria uma nova linha de execução na qual uma chamada *doIt* será executada, encarregando-se de anunciar o retorno da chamada ao invocador do serviço através da invocação do método *receiveServiceResponse*.

- ***receiveServiceResponse***: recebe a resposta de uma invocação de serviço assíncrono realizada por *doItAsynchronous*.

  Container: este método não é implementado por contêineres.

  FunctionalComponent: este método é implementado pelo desenvolvedor do componente.

5.3 Implementação do modelo de interação baseada em eventos

A implementação do modelo de interação baseada em eventos descrito na Seção 4.3.2 é fundamentada nos padrões *Observer* [GHJV95] e *Future* [LS95]. O algoritmo funciona com base na invocação do método *announceEvent* para anunciar o evento aos contêineres-pai dos componentes e na invocação do método *receiveEvent* para notificar o evento aos componentes interessados, seguindo a especificação descrita na Seção 4.3.2. A seguir é apresentada uma descrição das funcionalidades destes métodos assim como suas respectivas implementações para as classes *Container* e *FunctionalComponent*.

- ***announceEvent***: anuncia um evento aos outros componentes do sistema.

  Container: invoca o método *receiveEvent* e solicita ao seu contêiner-pai a invocação do método *announceEvent*

  FunctionalComponent: cria uma nova linha de execução na qual é feita uma solicitação ao seu contêiner-pai para a execução do método *announceEvent*.

- ***receiveEvent***: recebe um evento anunciado por um componente do sistema.

  Container: recebe o evento e o encaminha aos seus componentes-filhos de acordo com a tabela de eventos de interesse, exceto para o componente de origem do evento.

FunctionalComponent: recebe o evento e invoca o método definido pelo desenvolvedor do componente para tratar este evento.

5.4 Parâmetros de inicialização e Inicialização de componentes

Seguindo o modelo de especificação de novos componentes apresentado na Seção 4.1, o arcabouço COMPOR-F permite a definição de parâmetros de inicialização e provê mecanismos para a inicialização e a interrupção da execução dos componentes de um sistema. Os parâmetros de inicialização são armazenados em tabelas contidas nos componentes funcionais e podem ser acessados através do método *getInitializationParameter* que tem como argumento o nome do parâmetro de inicialização requerido. O mecanismo de inicialização/interrupção de componentes é implementado pelos métodos *start* e *stop*. Estes métodos, assim como suas respectivas implementações para as classes *Container* e *FunctionalComponent*, são descritos a seguir.

- ***start***: invocado no momento da inicialização dos componentes.

  Container: inicializa todos os componentes nele contidos através da invocação do método *start()*.

  FunctionalComponent: representa um *Template Method* [GHJV95] que inicializa as propriedades do componente e invoca um método implementado pelo desenvolvedor do componente para executar algum outro procedimento de inicialização.

- ***stop***: o funcionamento deste método é semelhante ao do método *start*, porém é invocado quando se deseja interromper a execução de um componente.

5.5 Especializando COMPOR-F para a arquitetura de agentes COMPOR

Como citado anteriormente, a motivação para a criação do modelo de componentes COMPOR-CM e, conseqüentemente, do arcabouço COMPOR-F é a necessidade de flexi-

bilizar o projeto dos agentes da arquitetura COMPOR, componentizando as suas funcionalidades. Utilizando o modelo COMPOR-CM para representar um agente da arquitetura COMPOR tem-se o seguinte resultado.

- Um **agente** é um **contêiner**.
- Um agente possui três outros **contêineres** que representam os sistemas **inteligente**, **social** e de **distribuição**.
- Cada sistema possui **componentes funcionais** que implementam as funcionalidades do agente.

Seguindo esta arquitetura, cada sistema possui componentes provedores das funcionalidades do agente. A comunicação entre dois agentes, por exemplo, acontece através da comunicação entre dois componentes funcionais que pertencem ao Sistema de Distribuição, como apresentado na Figura 5.2. Desta forma, obtém-se flexibilidade quanto à tecnologia de comunicação, protocolos e linguagens de interação utilizados pelos agentes, pois suas implementações estão encapsuladas nos componentes funcionais. Estes componentes funcionais, por sua vez, não têm referências entre si, flexibilizando assim os projetos do agente e das organizações.

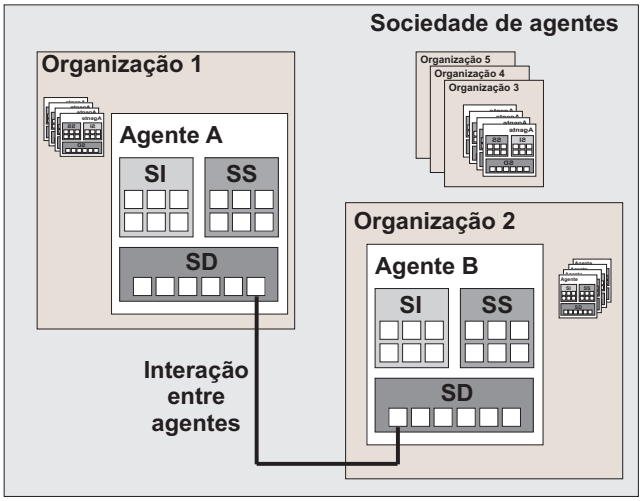

Figura 5.2: Arquitetura de agentes COMPOR: interação entre agentes através da interação entre componentes

Além da composição das entidades do modelo para formar um agente, é necessária a definição de uma seqüência de execução e das interações entre os componentes para que

o agente possa ser iniciado. O *script* de execução de um agente COMPOR é um componente chamado *Planejador*. Todo agente possui um planejador no seu sistema inteligente que determina a sua seqüencia de execução, sendo este componente sempre o último a ser iniciado e o primeiro a ser interrompido quando a inicialização ou interrupção de um agente é solicitada.

Como visto no Capítulo 3, as noções de sociedade e organização delimitam apenas o domínio a que os agentes pertencem e, sendo assim, não implementam a comunicação entre eles. A comunicação entre os agentes de uma organização, ou de organizações diferentes, acontece através dos componentes de comunicação dos agentes. Isto é necessário para permitir que, em uma determinada sociedade, agentes interajam utilizando diferentes tecnologias de comunicação, protocolos e linguagens, uma vez que o desenvolvedor de cada agente tem autonomia para definir os componentes que implementam estas tecnologias ao projetar seu agente.

Na Figura 5.3 é apresentado um cenário ilustrativo de comunicação [3] entre dois agentes que possuem suas arquiteturas baseadas no modelo COMPOR-CM. O *Agente A* possui um componente *Planejador*(P) e um componente de *Comunicação* (C). O *Agente B* possui, além do planejador que foi omitido nesta figura, um componente de *Soma*(S) e um componente de *Comunicação*(C). Os serviços de cada componente são descritos a seguir.

- **Componente de soma**: provê um serviço de soma de dois números inteiros.
  Descrição do serviço: "sum"
  Parâmetros: Integer aNumber, Integer otherNumber

- **Componente de comunicação**: provê a comunicação entre dois componentes via rede, repassando a solicitação de um determinado serviço.
  Descrição do serviço: "sendService"
  Parâmetros: ServiceRequest, onde ServiceRequest é um objeto que encapsula o nome de um serviço e seus parâmetros

O componente *Planejador*(P) do *Agente A* solicita o serviço de comunicação "sendService", enviando como parâmetro o serviço "sum" que será repassado para o *Agente B* através

[3] A intenção deste cenário é garantir o entendimento da comunicação entre dois agentes. A interação entre agentes em um sistema multiagentes demandaria a participação de componentes do sistema social.

do componente de comunicação. A seqüencia de passos ilustrados na Figura 5.3 é detalhada a seguir.

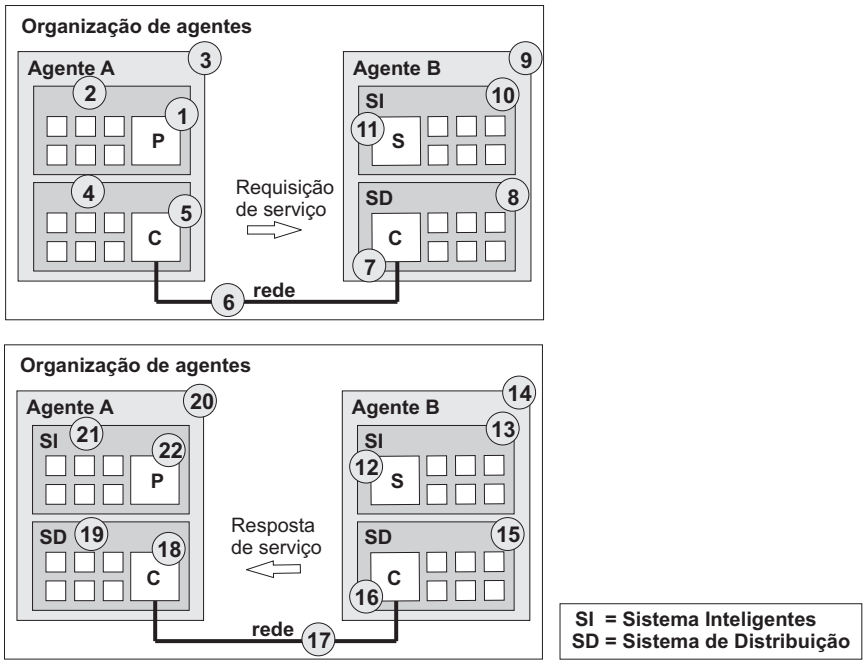

Figura 5.3: Comunicação entre componentes de agentes diferentes através de um componente de comunicação

1. O componente P do *Agente A* solicita a execução do serviço “sendService” através da invocação do método *doIt*, parametrizando o serviço “sum”: *doIt(sendRequest)*, onde *sendRequest* encapsula [“sendService”, [“sum”, [aNumber, otherNumber]]].

2. A solicitação chega ao SI do *Agente A* que verifica se algum de seus componentes-filhos implementa este serviço. Como não existe nenhum componente-filho em SI que implemente o serviço, então a execução do método *doIt* é encaminhada ao seu contêiner-pai, neste caso, *Agente A*.

3. O *Agente A*, por sua vez, verifica que o seu SD possui algum componente que implementa o serviço, então ele delega a execução do método *doIt* para ele.

4. O SD do *Agente A* encaminha a execução do método *doIt* ao componente C que implementa o serviço “sendService”.

5. O componente C do *Agente A* envia, através da rede, uma mensagem ao componente C do *Agente B*, contendo o serviço a ser executado remotamente (“sum”), bem como os parâmetros necessários para a sua execução (aNumber, otherNumber).

6. Como citado anteriormente, a interação entre os componentes via rede representa a comunicação entre os agentes da sociedade.

7. O componente C do *Agente B* recebe a mensagem e extrai o serviço e os parâmetros a serem solicitados neste agente e invoca o método *doIt*, com o serviço “sum” e seus parâmetros: *doIt(sumRequest)* onde *sumRequest* encapsula [“sum”, [aNumber,otherNumber]].

8. A execução do método *doIt* é encaminhada ao SD do *Agente B*.

9. O SD do *Agente B* o encaminha ao contêiner-pai (*Agente B*).

10. O contêiner *Agente B* encaminha a solicitação ao SI.

11. O SI do *Agente B* delega a execução ao seu componente-filho S, que por sua vez executa a soma e retorna o resultado.

Os passos 12-22, ilustrados na Figura 5.3, representam o retorno do resultado da invocação de serviço percorrendo o caminho inverso da solicitação: do servidor (S) ao cliente (P) do serviço, passando pelo componente de comunicação.

Uma vez apresentada a adequação do modelo COMPOR-CM à arquitetura de agentes COMPOR, na Figura 5.4 é apresentado o modelo de classes do arcabouço COMPOR-F especializado para a mesma arquitetura. De acordo com este modelo, as classes *Organisation* e *Society* não são extensões da classe *Container*. Isto se deve ao fato de que o modelo de componentes COMPOR-CM é aplicado apenas a cada agente, individualmente, ou seja, dentro de cada agente há uma hierarquia de componentes COMPOR-CM, como visto na Figura 5.3.

5.6 Construindo uma sociedade de agentes

Para a construção de uma sociedade de agentes é necessário definir que organizações fazem parte da sociedade, os agentes que compõem as organizações e, por fim, os componentes

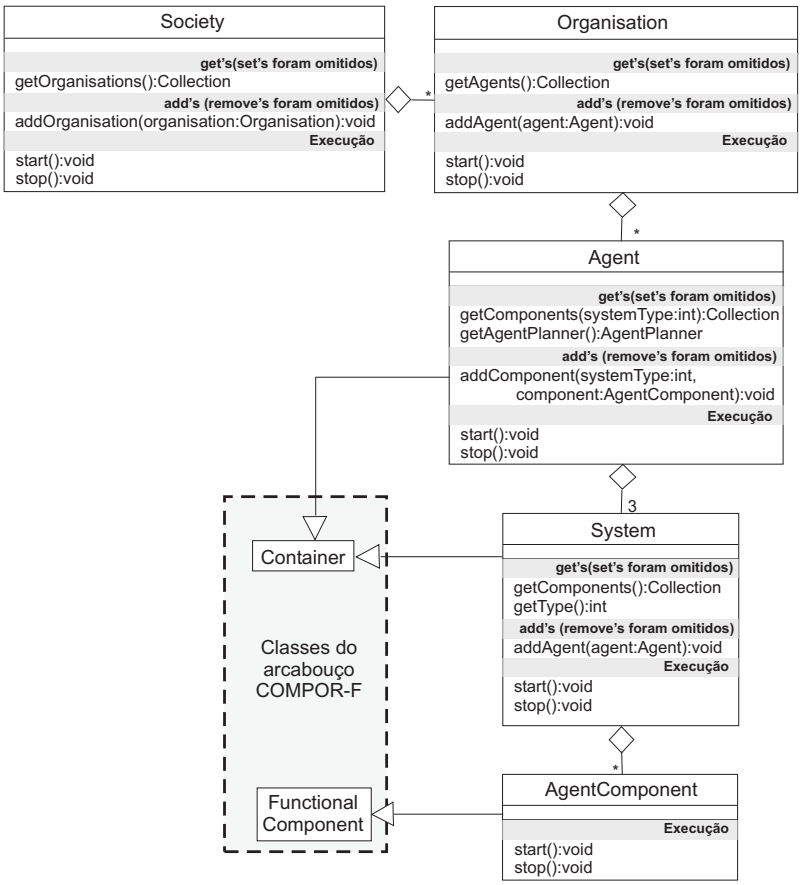

Figura 5.4: Diagrama de classes COMPOR-F aplicado à arquitetura de agentes COMPOR

que implementam as funcionalidades dos agentes. Considerando-se então que a sociedade de agentes já está definida em relação à análise e projeto apresentados, respectivamente, nas Seções 3.2.1 e 3.2.2, a seguir é apresentada a seqüencia de passos para o desenvolvimento de uma aplicação multiagentes utilizando o modelo apresentado na Figura 5.4.

Para esse exemplo, são utilizados os componentes do cenário ilustrado na Figura 5.3. O componente de soma é implementado pela classe *SumComponent* e o componente de comunicação é implementado pela classe *ComComponent*.

1. Definem-se a sociedade e as organizações que farão parte do sistema multiagentes:

```
Society sociedade = new Society(“MinhaSociedade”);
Organisation org = new Organisation(“Org1”);
sociedade.addOrganisation(org);
```

2. Definem-se os agentes que pertencem a cada organização:

```
Agent agenteA = new Agent(“Agente A”);
```

```
Agent agenteB = new Agent("Agente B");
org.addAgent(agenteA);
org.addAgent(agenteB);
```

3. Definem-se os componentes que pertencem aos agentes:

```
SumComp compSoma = new SumComp();
ComComp compCom = new ComComp();
agenteA.addComponent(System.DISTRIBUTION, compCom);
agenteB.addComponent(System.INTELLIGENT, compSoma);
agenteB.addComponent(System.DISTRIBUTION, compCom);
```

4. Definem-se os planejadores dos agentes. Na Figura 5.5 apresenta-se o código do planejador para este exemplo [4]:

```
agenteA.setAgentPlanner(new RemoteSumPlanner());
```

```
public void init{

    /* Define os parâmetros do serviço "sum"
    Integer parcela1 = new Integer(5);
    Integer parcela2 = new Integer(10);
    Object[] pSoma = new Object[]{parcela1, parcela2};

    /* Define a requisição de soma */
    ServiceRequest rqSoma = new ServiceRequest("sum", pSoma);

    /* Define os parâmetros do serviço "sendRequest" */
    Object[] pEnv = new Object[]\{"192.168.7.10", pSoma\};

    /* Define a requisição do serviço "sendRequest" */
    ServiceRequest rqEnv = new ServiceRequest("sendService", pEnv);

    /* Invoca o serviço de forma síncrona */
    ServiceResponse resposta = this.doIt(rqEnv);

    /* Imprime a saída */
    Integer resultadoSoma = (Integer)resposta.getData();\\
    System.out.print(resultadoSoma.intValue());
}

public void finish{}
```

Figura 5.5: Exemplo de código de planejador de agente

5. Inicializa-se a sociedade. Isto acarretará a inicialização de todas as organizações, agentes e componentes do sistema, inclusive do planejador definido acima.

[4] O planejador do *AgenteB* foi omitido pois, neste exemplo, este agente é responsável apenas por receber as requisições dos serviços de soma.

```
sociedade.start();
```

O processo de criação e montagem dos componentes, dos agentes e da sociedade de agentes do exemplo anterior foi apresentado para um melhor entendimento do leitor em relação ao funcionamento de um sistema multiagentes utilizando o arcabouço COMPOR-F. Porém, a quantidade de componentes, agentes e organizações utilizadas não condiz com a realidade de implementação de um sistema. A complexidade de um sistema multiagentes torna inviável o desenvolvimento sem o auxílio de uma ferramenta. Por isso, apresenta-se no Capítulo 6 um ambiente de desenvolvimento que automatiza o processo de criação de sociedades, organizações, agentes e componentes de acordo com as diretrizes metodológicas COMPOR-M, modelo de componentes COMPOR-CM e arcabouço COMPOR-F.

5.7 Arcabouços correlatos

Alguns arcabouços orientados a objetos para a construção de sistemas multiagentes foram propostos. Embora a maioria deles implemente a infra-estrutura necessária para a interação entre os agentes, há pouca flexibilidade para a implementação de novas tecnologias, linguagens ou protocolos para esta interação. Isto acontece porque a implementação das funcionalidades dos agentes não está encapsulada em componentes e, desta forma, não podem ser redefinidas com facilidade. Esta característica pode ser observada nas soluções propostas em [ZA00; BFP00; AA00; Cha97; MCM99; GSP98], as quais são analisadas e comparadas ao arcabouço COMPOR-F no Apêndice C. Nestes arcabouços, o projeto dos agentes é dependente das funcionalidades específicas providas por estes agentes, dificultando a reutilização e mudança nas funcionalidades.

O arcabouço JAF [VHL01] é baseado em componentes e possui uma estrutura de agentes semelhante a COMPOR-F - agentes formados por componentes. Porém o modelo de componentes no qual JAF se baseia é *JavaBeans* que, como apresentado no Capítulo 4, não possui independência entre os serviços disponibilizados e requeridos pelos componentes funcionais. O arcabouço COMPOR-F provê interação assíncrona e seu modelo de classes é suficientemente simples para ser implementado em outras linguagens. Isto não ocorre em relação a arcabouços que possuem como base modelos de componentes como *JavaBeans*.

Capítulo 6

Ambiente de desenvolvimento - COMPOR-E

Para automatizar o processo de desenvolvimento e tornar mais eficiente a utilização do arcabouço apresentado no Capítulo 5, um ambiente CASE [1] foi construído. O ambiente COMPOR-E [2] [ALF+03] possui um conjunto de ferramentas para auxiliar o desenvolvedor nas diversas fases de concepção de um software utilizando a abordagem multiagentes. Para isso, o ambiente se baseia nas diretrizes definidas no Capítulo 3, dando suporte às fases de análise, projeto, implementação, testes e gerência de execução como apresentado na Figura 6.1

A arquitetura do ambiente COMPOR-E e das ferramentas que o compõem são descritas nas seções seguintes.

6.1 Arquitetura do ambiente

A arquitetura interna do ambiente é ilustrada na Figura 6.2. O arcabouço COMPOR-E é implementado pelo ambiente e disponibilizado através da API do Ambiente, apresentada nesta figura. Esta API encontra-se disponível para ser acessada pelas ferramentas. O projeto de agentes também é acessado e modificado a partir da API. A troca de informações entre as

[1] Um ambiente CASE(*Computer Aided Software Engineering*) representa um conjunto de ferramentas que auxiliam no processo de desenvolvimento de software [Pre95]

[2] Abreviação para COMPOR-*Environment*

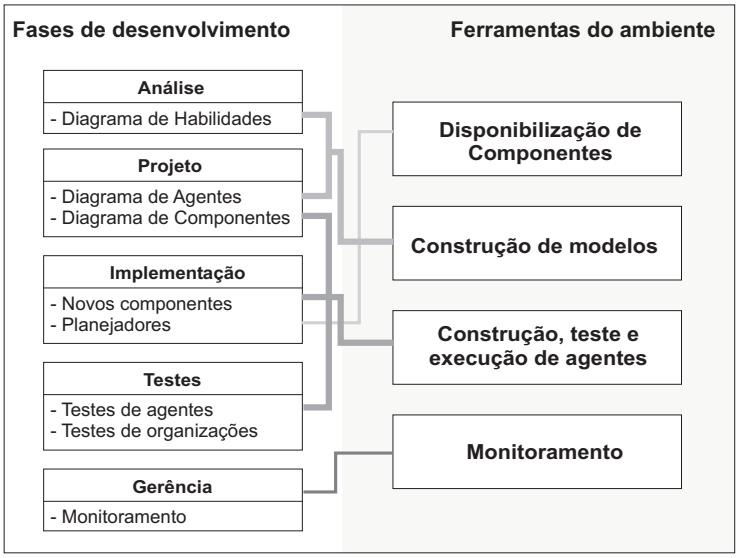

Figura 6.1: Fases de desenvolvimento contempladas por COMPOR-E

ferramentas que compõem o ambiente acontece através de eventos. O padrão de projeto *Observer* [GHJV95] foi utilizado para tornar as ferramentas independentes do ambiente e umas das outras. Sendo assim, quando um evento é disparado por uma determinada ferramenta, a API do ambiente é notificada e encaminha o evento para as outras ferramentas. A interface das ferramentas é acoplada ao ambiente também através da API, sendo então disponibilizada para o usuário. O ambiente dá suporte à internacionalização [3] através de *properties* [4], cuja implementação é provida pela própria API de Java.

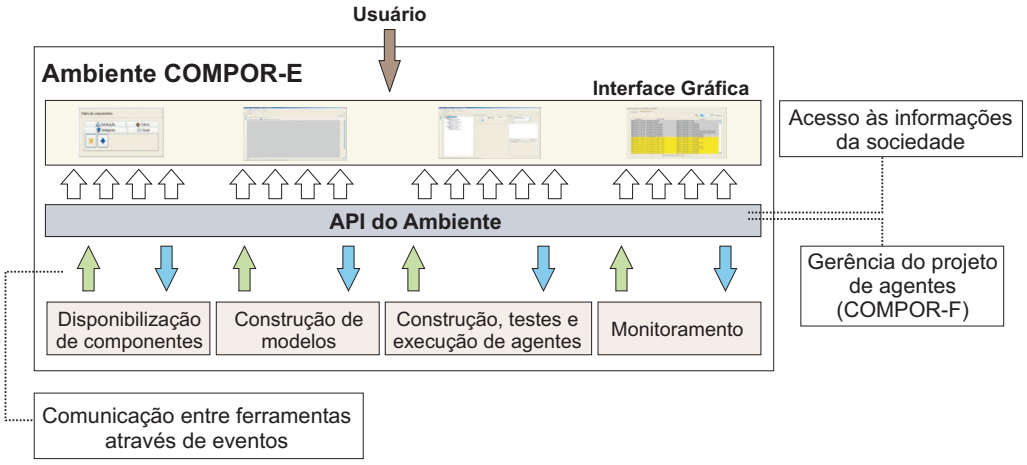

Figura 6.2: Arquitetura interna do ambiente COMPOR-E

[3] Ferramenta para facilitar a utilização do software em vários países e idiomas

[4] Properties Java:http://java.sun.com/docs/books/tutorial/essential/attributes/properties.html

A interface do ambiente é baseada no conceito de perspectivas, inspirado no Projeto Eclipse [Ecl03]. Cada perspectiva diz respeito a uma determinada partição do ambiente que contém ferramentas relativas a um contexto específico. O ambiente contempla quatro perspectivas principais, das quais três se encontram implementadas atualmente.

- **Perspectiva de engenharia do conhecimento:** nesta perspectiva deverão ser inseridas as ferramentas de edição e manutenção de conhecimento, mais especificamente de ontologias. Esta perspectiva ainda não foi desenvolvida.

- **Perspectiva de construção de modelos:** nesta perspectiva, ilustrada na Figura 6.3, estão contidas as ferramentas relacionadas à modelagem da sociedade de agentes de acordo com as diretrizes COMPOR-M.

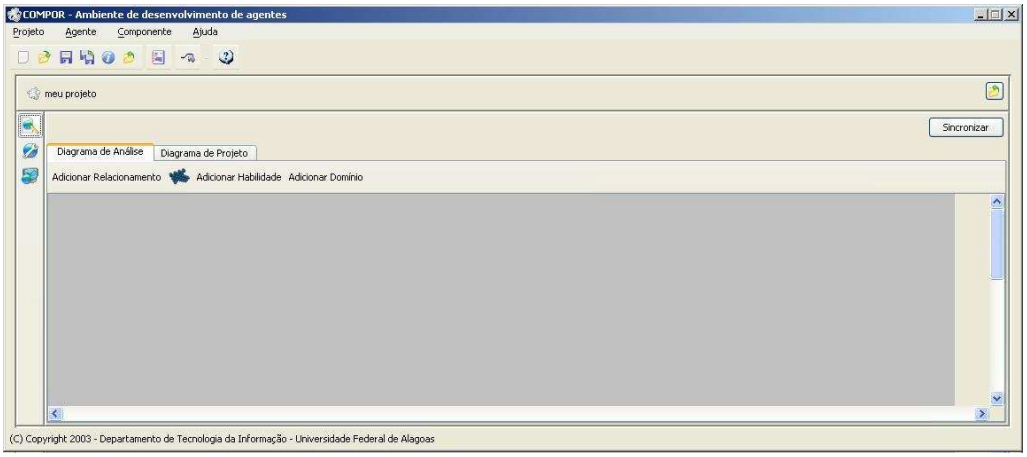

Figura 6.3: Perspectiva de construção de modelos

- **Perspectiva de construção de agentes:** as ferramentas relacionadas ao projeto, implementação, testes e execução de agentes estão contidas nesta perspectiva, ilustrada na Figura 6.4.

- **Perspectiva de monitoramento:** as ferramentas de monitoramento da execução dos agentes estão contidas nesta perspectiva, ilustrada na Figura 6.5.

O ambiente é composto por quatro ferramentas principais que são descritas a seguir. De acordo com o contexto em que se aplicam e suas nomenclaturas, fica claro o mapeamento para as perspectivas descritas acima.

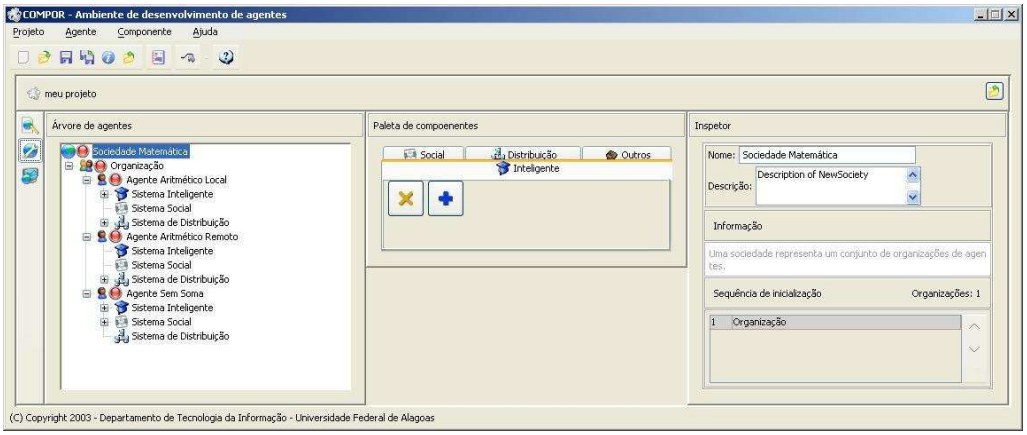

Figura 6.4: Perspectiva de construção de agentes

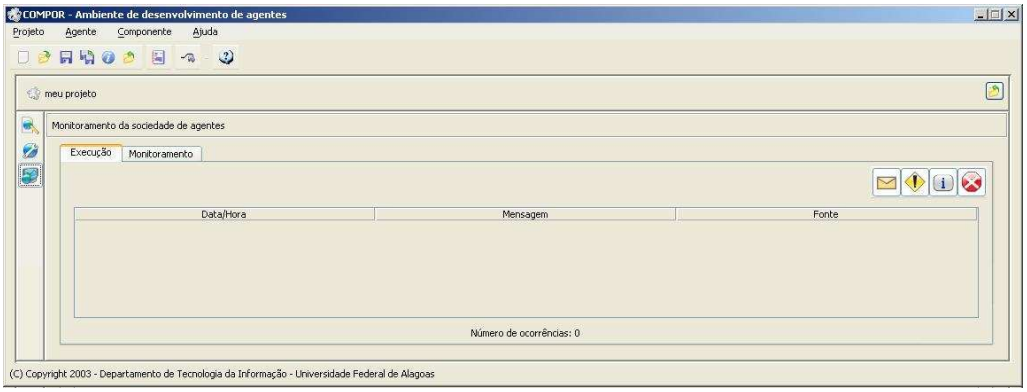

Figura 6.5: Perspectiva de monitoramento

6.1.1 Ferramenta de disponibilização de componentes

Utilizada para a disponibilização de novos componentes ao ambiente. Os componentes devem seguir a especificação do modelo de componentes COMPOR-CM, apresentado no Capítulo 4. Para garantir o cumprimento da especificação, a adição de componentes ao ambiente é feita através de um formulário "passo a passo" (*wizard*) que recebe como entrada os dados da especificação do componente. Na Figura 6.6 é apresentado o formulário de adição de novos componentes à ferramenta.

Os componentes adicionados são dispostos em uma paleta de componentes, ilustrada na Figura 6.7, que representa a biblioteca de componentes COMPOR. Quanto mais rica for esta biblioteca menor será o esforço de implementação do desenvolvedor de um sistema [Crn01]. A paleta possui uma divisão contextual de acordo com o modelo COMPOR-CM, classificando os componentes como Inteligente, Social, Distribuição e Outros.

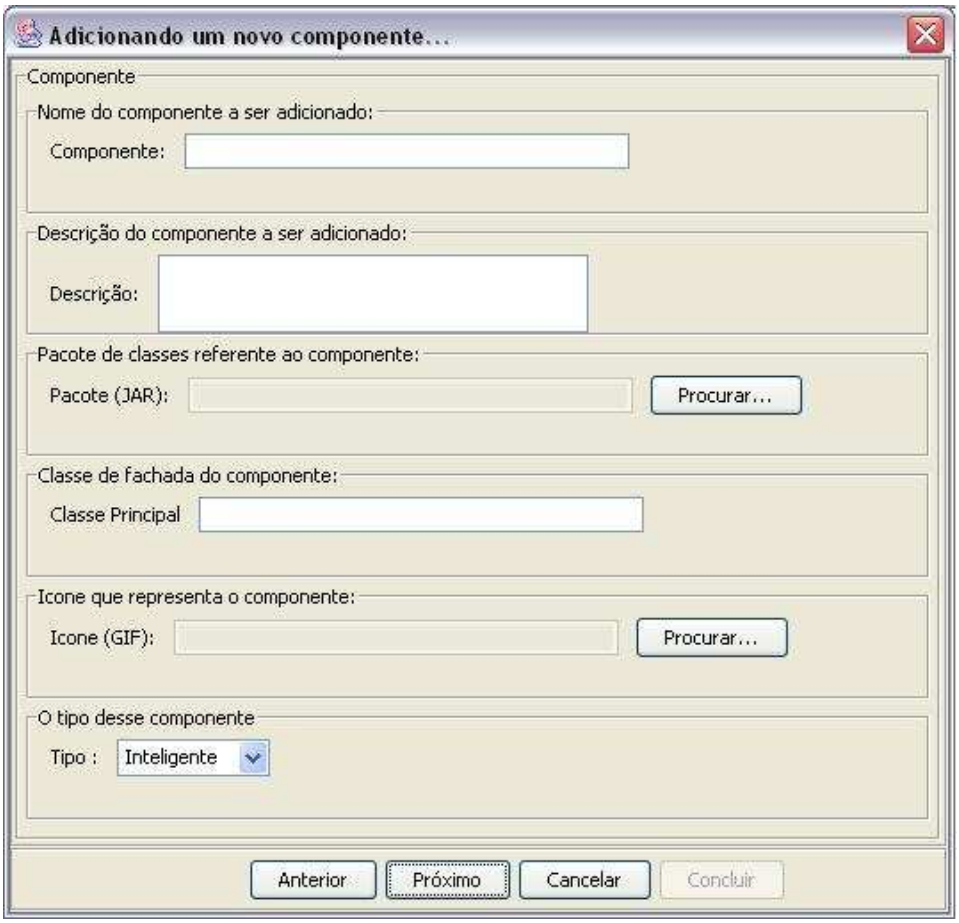

Figura 6.6: *Wizard* para adição de novos componentes

O desenvolvimento de componentes para a biblioteca COMPOR está em andamento, com a construção de componentes de comunicação entre agentes, mecanismos de inferência e recuperação de informação. A ferramenta aqui proposta provê a infra-estrutura necessária à componentização das funcionalidades, garantindo flexibilidade na concepção de sistemas multiagentes. O enriquecimento da biblioteca virá com a utilização do ambiente em domínios de aplicação diversos.

6.1.2 Ferramenta de modelagem

Utilizada para auxiliar o desenvolvedor na aplicação das diretrizes apresentadas no Capítulo 3. Possui módulos de construção de diagramas e geração automatizada de projetos baseada nos diagramas.

A geração de projetos é feita através do mapeamento de diagramas em componentes

Figura 6.7: Biblioteca de componentes COMPOR

e agentes do sistema. Os modelos gerados por esta ferramenta são sincronizados com a arquitetura de agentes gerenciada pela ferramenta de construção de agentes, através do armazenamento de informações no formato XML.

O desenvolvimento desta ferramenta está em andamento, no contexto de um trabalho de conclusão de curso na Universidade Federal de Alagoas.

6.1.3 Ferramenta de construção, teste e execução de agentes

Utilizada para a construção da sociedade de agentes. Na Figura 6.8 é apresentada a árvore de agentes do sistema, onde é possível adicionar novas organizações e agentes, bem como especificar por quais componentes cada agente é composto através do *Drag and Drop* dos componentes que estão na paleta de componentes.

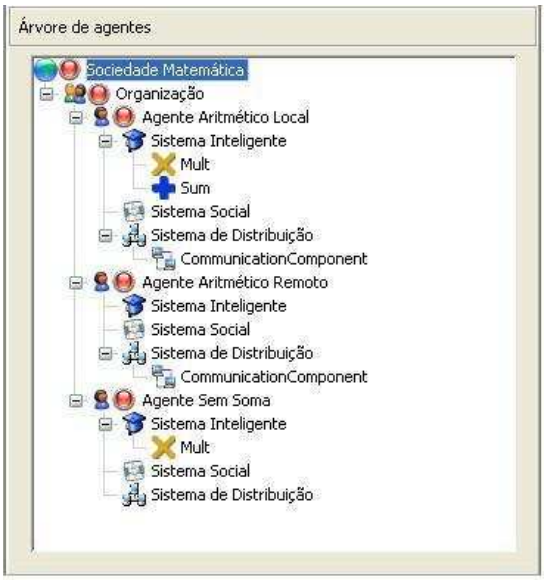

Figura 6.8: Árvore de construção de agentes

Além da árvore de agentes, a ferramenta possui um “inspetor” dos nós da árvore, possibilitando a visualização e alteração das suas propriedades. Na Figura 6.9 é apresentado o estado do inspetor para a árvore da Figura 6.8.

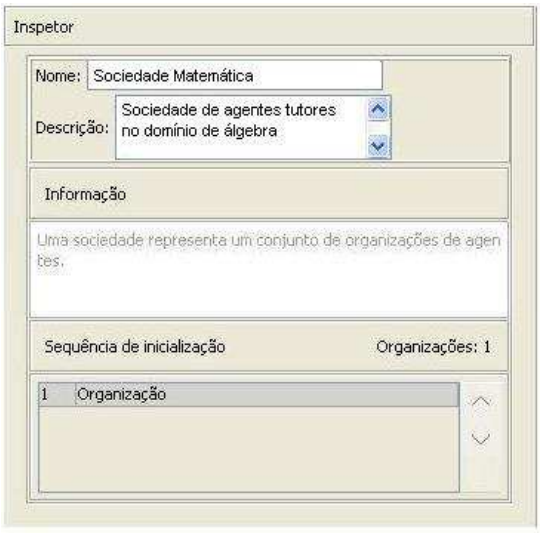

Figura 6.9: Inspetor da construção de agentes

Para cada nó selecionado na árvore de agentes há uma diferente configuração no inspetor. Quando o nó raiz “Sociedade”, os nós “Organização” e os nós “Agente” são selecionados, por exemplo, o inspetor exibe uma tabela com a seqüência de inicialização das organizações, agentes e componentes, respectivamente. Através desta tabela é possível alterar a seqüência de inicialização do sistema. Além disso, informações sobre o nó selecionado são exibidas com detalhes através do inspetor como, por exemplo, o código fonte dos planejadores dos agentes. Na Figura 6.10 é apresentado o estado do inspetor ao selecionar-se um nó “Agente”.

No caso dos componentes, o inspetor é uma ferramenta indispensável porque exibe as informações sobre os serviços providos, serviços requeridos, eventos disparados e eventos de interesse. Estas informações são necessárias para que sejam verificadas as dependências entre os serviços e, conseqüentemente, entre os componentes no momento da “montagem” de um agente. Além disso, parâmetros de inicialização dos componentes também são definidos através do inspetor. Na Figura 6.11 é apresentado o estado do inspetor ao selecionar-se um nó “Componente”.

Ao pressionar o botão direito do mouse sobre um nó da árvore, visualiza-se um *menu popup* através do qual pode-se testar, inicializar e interromper a execução dos agentes. A

Figura 6.10: Inspetor exibindo informações sobre um nó “Agente”

inicialização seguirá a seqüência definida e apresentada no inspetor, da mesma forma que a interrupção seguirá a seqüência inversa. Também através deste menu, o planejador de um agente é definido via seleção do arquivo de classes que implementa o código deste planejador. Uma caixa de diálogo para a seleção do planejador é exibida.

Os testes também são realizados através do *menu popup* da árvore, com auxílio do arcabouço de testes de unidade JUnit[BG98]. Esta ferramenta, identifica todas as dependências entre os componentes adicionados aos agentes e identificar se existem dependências entre componentes que não estão sendo contempladas no projeto de agentes. Isto é feito através da implementação de casos de testes no código do arcabouço. Na Figura 6.12 é apresentada a interface do arcabouço JUnit acoplada ao ambiente COMPOR-E e adaptada para testes de dependência entre componentes.

Os testes funcionais podem ser executados em cada componente, agente ou na sociedade como um todo. Os testes dizem respeito à dependência entre componentes e entre agentes.

Com o auxílio desta ferramenta o desenvolvedor tem suporte a todas as diretrizes de projeto, implementação e testes, com exceção das diretrizes para a construção de diagramas, as quais se referem à ferramenta de construção de modelos.

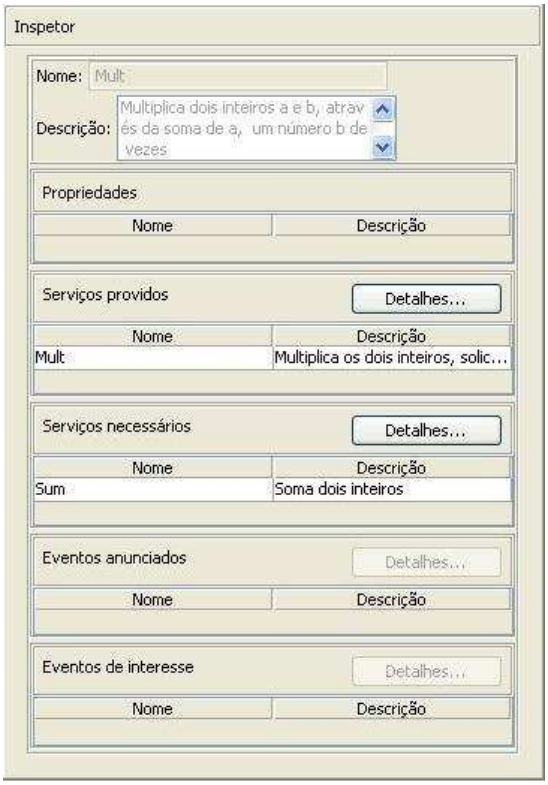

Figura 6.11: Inspetor exibindo informações sobre um nó “Componente”

6.1.4 Ferramenta de monitoramento

Utilizada para o controle e gerenciamento da execução dos agentes. O monitoramento é feito através da interceptação de mensagens enviadas entre agentes e entre componentes do mesmo agente. A ferramenta de monitoramento classifica as mensagens interceptadas em dois grandes grupos, os quais são exibidos em tabelas diferentes na interface do ambiente.

O primeiro grupo contém as mensagens geradas pelos desenvolvedores dos componentes que, de acordo com a necessidade, enviaram mensagens através do arcabouço COMPOR-F. Supondo, por exemplo, que o desenvolvedor de um componente de comunicação desejasse registrar o IP dos seus clientes a cada conexão ou requisição de serviços. Sendo este componente uma instância de *AgentComponent*, como visto no Capítulo 5, este possui acesso a um método da sua classe-pai chamado *outConsole()*. Através deste método, é possível disparar mensagens de registro ou log caso o desenvolvedor considere pertinente, classificando-as como mensagens de Alerta, Erro ou Informação. São estas as mensagens apresentadas na interface do ambiente COMPOR-E, como apresentado na Figura 6.13. Os ícones na parte superior determinam um filtro para as mensagens, subdividindo-as em mensagens de Alerta,

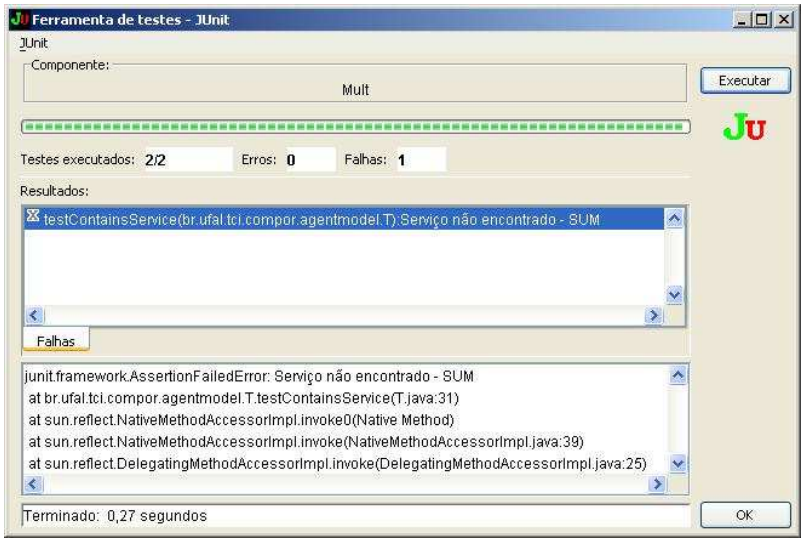

Figura 6.12: Ferramenta JUnit adaptada para testes de dependências

Erro e Informação.

Figura 6.13: Monitoramento de mensagens geradas pelos desenvolvedores dos componentes

O segundo grupo contém as mensagens geradas pelo arcabouço COMPOR-F em relação à invocação de serviços, adição de agentes e componentes, inicialização e interrupção de agentes e componentes, dentre outros. Através desta tabela pode-se visualizar todos os eventos ocorridos durante a execução e construção do sistema, facilitando a manutenção e verificação de erros. Na Figura 6.14 é apresentada a interface de monitoramento de eventos. Os ícones na parte superior determinam um filtro para os eventos, subdividindo-os em eventos de Construção, Inicialização/Interrupção e Execução de serviços. Além disso é possível configurar os filtros através da interface, como ilustrado na Figura 6.15.

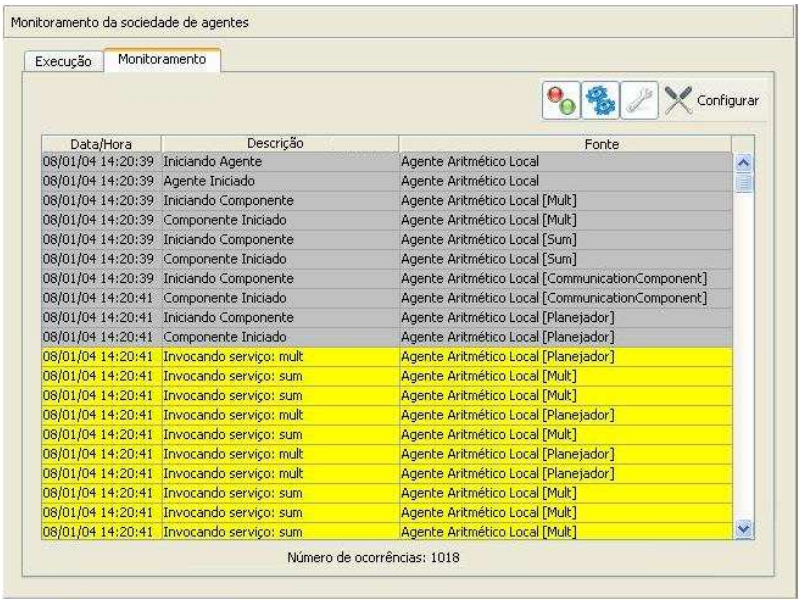

Figura 6.14: Monitoramento de eventos gerados pelo arcabouço COMPOR-F

A ferramenta de monitoramento é essencial para a verificação do funcionamento correto do sistema. Uma vez que a complexidade de gerenciamento cresce proporcionalmente ao número de agentes e componentes pertencentes à sociedade, é indispensável ao desenvolvedor uma forma de monitorar visualmente o estado de seu sistema.

6.2 Ferramentas e ambientes correlatos

Algumas ferramentas para o desenvolvimento de aplicações multiagentes foram analisadas como candidatas a prover a infra-estrutura para a aplicação das diretrizes COMPOR-M. Algumas possuem arcabouços ou metodologias associadas como é o caso da ferramenta JADE [BRP99], que possui um arcabouço de software de mesmo nome, e agentTool [Lab03], que está associada à metodologia MaSE [DWS01]. Outras possuem funcionalidades voltadas para a utilização de técnicas de inteligência artificial como Zeus [Zeu03] e MadKIT [GFM03], tendo pouca flexibilidade para a construção de software utilizando a entidade Agente como abstração.

De um modo geral, as ferramentas contemplam um projeto de Agente extremamente acoplado às suas funcionalidades. Esta característica é similar àquela encontrada nos arcabouços correlatos a COMPOR-F, como descrito no Capítulo 5. Sendo assim, a motivação para

Figura 6.15: Configuração de filtros de eventos

a construção de COMPOR-E vem da necessidade de automatizar a aplicação de COMPOR-F, uma vez que COMPOR-F provê maior flexibilidade que os arcabouços “automatizados” pelas outras ferramentas.

No Apêndice D é apresentada uma análise das ferramentas relacionadas, com ênfase nas contribuições de COMPOR-E em relação às mesmas.

Capítulo 7

Estudo de caso - Empresa virtual

Para ilustrar a aplicabilidade da infra-estrutura de Engenharia de Software proposta neste trabalho, apresenta-se nesta seção a utilização do COMPOR no desenvolvimento de um sistema de empresa virtual. O foco deste capítulo é apresentar como a infra-estrutura COMPOR pode ser utilizada no desenvolvimento de um sistema de software complexo, nos termos definidos nesta dissertação.

7.1 Empresa virtual - Um problema modelo

Uma empresa virtual é definida como uma organização que repassa a terceiros parte de suas atividades para outras empresas(parceiras), integrando seus sistemas de informação aos destas empresas [Ver96]. Exemplos de situações complexas neste tipo de sistema são descritas a seguir, conforme características mencionadas no Capítulo 2.

- O resultado da integração dos sistemas de informação pode ser visto como um novo sistema, que possui um conjunto de subsistemas autônomos interagindo.
- Os subsistemas podem estar espacialmente distribuídos e terem sido implementados utilizando linguagens e plataformas diferentes.
- Os domínios de conhecimento inerentes a cada sistema são provavelmente diferentes.
- As empresas parceiras não são fixas. A empresa pode, e deve, mudar suas parcerias em prol de um melhor benefício próprio. Da mesma forma, novas parcerias podem ser formadas e novos produtos e serviços serem disponibilizados.

Como as características listadas para um sistema de empresa virtual são independentes do serviço negociado pela empresa e do tipo de parceria, um tal sistema pode ser assim caracterizado como representante de uma ampla classe de sistemas, com propriedades semelhantes, onde se pode localizar, por exemplo, domínios como: comércio eletrônico [FCA03], cadeia de suprimentos [WW99], gestão de conhecimento [HSAG03a] etc. Dada a abrangência deste tipo de sistema é que ele foi escolhido como estudo de caso de aplicação da infra-estrutura COMPOR.

7.2 *BonVoyage* - Empresa virtual de planejamento de viagens

Nesta seção é descrito o cenário de funcionamento de uma empresa virtual, aqui representada por um sistema de planejamento de viagens chamada *BonVoyage* [APCP04]. O principal produto desta empresa são pacotes de viagem formulados e vendidos através da Internet. Para isto, solicita-se ao cliente a definição dos locais de origem e destino e a partir destes dados são fornecidas:

- opções de transporte a este local, que podem ser avião, trem ou ônibus;
- opções de hospedagem disponíveis no local de destino, que podem ser hotéis, pousadas e albergues;
- opções de lazer disponíveis no local de destino, que podem ser passeios turísticos e ingressos para apresentações artísticas.

Para obter todas estas informações a empresa mantém parcerias com uma série de prestadoras de serviços, as quais concorrem para negociar com a *BonVoyage*. Estes relacionamentos com as empresas parceiras são transparentes para o cliente. Além disso, estas empresas funcionam independentemente da *BonVoyage* e possuem autonomia para mudar seus preços, pesquisar os preços da concorrência, negar a prestação de serviços, mudar a forma como os seus serviços são oferecidos etc. Na Figura 7.1 são ilustradas as parcerias da empresa *BonVoyage*.

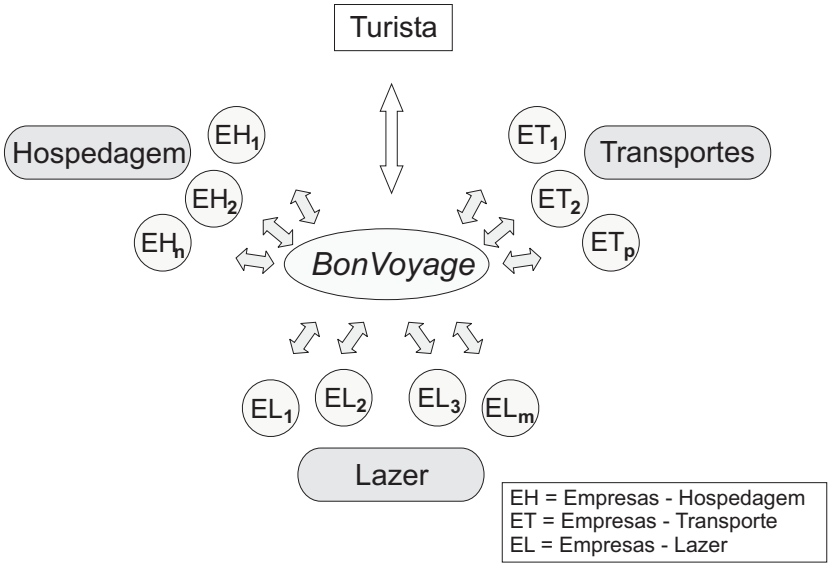

Figura 7.1: Parcerias da empresa virtual *BonVoyage*

Devido ao relacionamento com empresas que atuam em ramos de atividade distintos, a *BonVoyage* precisa ter acesso aos vários domínios de conhecimento referentes a estes ramos de atividade, para que possa negociar com estas empresas e repassar as informações sobre os serviços negociados ao cliente. Por exemplo, é necessário entender como funciona o transporte de avião para mostrar aos seus clientes atributos como localização da poltrona, escalas, conexões, tempo de vôo etc.

Em se tratando de competitividade, é natural a constante evolução da empresa para se adaptar às necessidades do mercado. Esta evolução pode-se refletir em:

- novos serviços prestados pelas empresas parceiras existentes;
- novas parcerias formadas, possivelmente disponibilizando novos serviços a serem negociados;
- mudanças nos processos de negociação com as empresas parceiras;

7.3 Arquitetura multiagentes para a empresa virtual

Para ilustrar a adequação da abordagem multiagentes no desenvolvimento de sistemas complexos, nesta seção é apresentada uma arquitetura multiagentes para dar suporte à empresa

BonVoyage. Como visto anteriormente, cada empresa parceira pode ser vista como um subsistema que compõe o sistema da empresa virtual. Cada um dos subsistemas possui autonomia e a interação entre eles possui características de distribuição e heterogeneidade. Além disso, os domínios de conhecimento inerentes a cada empresa são distintos. Sendo assim, a abstração multiagentes, conforme descrito no Capítulo 2, demonstra-se adequada para representá-los.

Uma possível arquitetura multiagentes para este sistema é apresentada na Figura 7.2. Nesta figura, ilustra-se a interação do turista através de um navegador Web com o agente representante da empresa *BonVoyage*. Este agente recebe as requisições do usuário e interage com os agentes das empresas parceiras para definir os pacotes de viagens. Esta interação se dá através das linguagens de comunicação entre agentes, tais como FIPA-ACL que disponibiliza um conjunto de performativas para a definição de protocolos de interação.

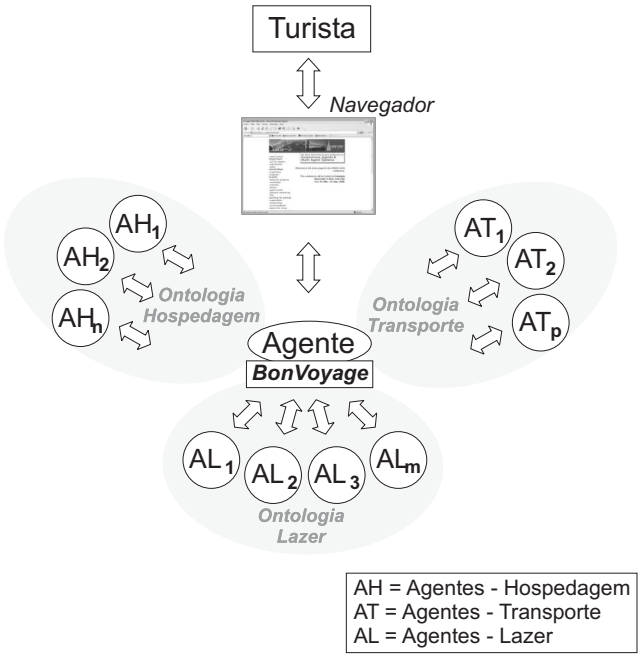

Figura 7.2: Arquitetura de agentes para a empresa *BonVoyage*

Cada parceira da empresa *BonVoyage* é mapeada em agentes os quais compartilham a ontologia referente ao domínio a que dizem respeito. Sendo assim, agentes relacionados ao domínio de transporte compartilham a ontologia que define os conceitos e seus relacionamentos em relação a este domínio. O mesmo acontece para os domínios de lazer e hospedagem. Desta forma, assegura-se que não há ambigüidades de conceitos apesar da

heterogeneidade dos subsistemas, representados pelos agentes.

7.4 Identificando os contextos e habilidades

Com base nas diretrizes COMPOR-M apresentadas no Capítulo 3, podem ser definidos os contextos do domínio do problema de planejamento de viagens e as habilidades referentes a cada contexto. Também de acordo com a Figura 7.2, pode-se definir, ou reutilizar, uma ontologia de Turismo/Planejamento de viagens. A definição da ontologia a ser utilizada está fora do escopo deste trabalho. Considere-se então que uma ontologia para o domínio de turismo foi utilizada para descrever o conhecimento envolvido no sistema e partições desta ontologia foram definidas para cada um dos contextos listados abaixo e representados no diagrama da Figura 7.3. O contexto de Planejamento estaria relacionado à versão completa da ontologia de turismo, uma vez que as habilidades referentes a este contexto lidam com conceitos dos três tipos de serviços disponibilizados: hospedagem, transporte e lazer.

Habilidades no contexto de Hospedagem

1. **Fornecer opções de hospedagem -** Fornecer opções de hospedagem disponíveis de acordo com parâmetros de configuração do serviço.
2. **Negociar serviço de hospedagem -** Negociar determinado serviço de hospedagem.

Habilidades no contexto de Transporte

1. **Fornecer opções de transporte -** Fornecer opções de transporte disponíveis de acordo com parâmetros de configuração do serviço.
2. **Negociar serviço de transporte -** Negociar determinado serviço de transporte.

Habilidades no contexto de Lazer

1. **Fornecer opções de lazer -** Fornecer opções de lazer disponíveis de acordo com parâmetros de configuração do serviço.
2. **Negociar serviço de lazer -** Negociar determinado serviço de lazer.

Habilidades no contexto de Planejamento

1. **Definir pacote de viagem -** Definir as opções de pacotes de viagem disponíveis de acordo com parâmetros de configuração do serviço.

2. **Negociar serviço com consumidor -** Negociar determinado serviço com o consumidor.

3. **Negociar serviço com parceiros -** Negociar determinado serviço com parceiros.

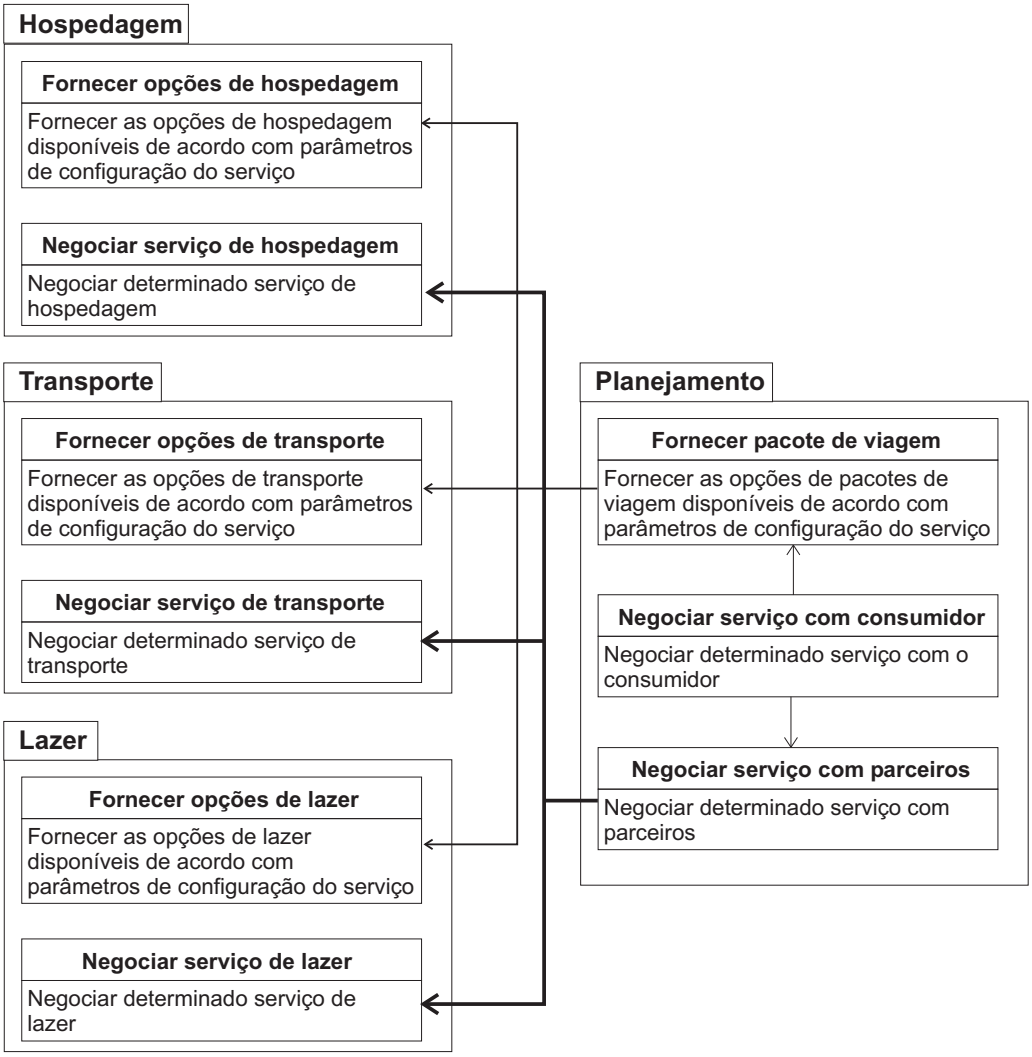

Figura 7.3: Diagrama de habilidades para o sistema da empresa *BonVoyage*

As dependências entre as habilidades também são representadas no diagrama, como descrito no Apêndice A. No exemplo da empresa *BonVoyage*, as habilidades vinculadas ao

contexto de planejamento são dependentes das habilidades dos outros contextos, como ilustrado na Figura 7.3.

7.5 Definindo as organizações e os agentes

Uma vez definidos os contextos e habilidades relacionados ao sistema da empresa *BonVoyage*, de acordo com as diretrizes COMPOR-M, deve-se mapear os contextos para organizações de agentes e alocar as habilidades a agentes destas organizações. A definição dos agentes é dependente da aplicação e pode ser influencidada por requisitos não-funcionais, como desempenho ou robustez, por exemplo. No caso da *BonVoyage* será definido um agente como representativo de cada organização, para simplificar os diagramas. Na Figura 7.4 é apresentado o diagrama de agentes para o sistema da empresa *BonVoyage*.

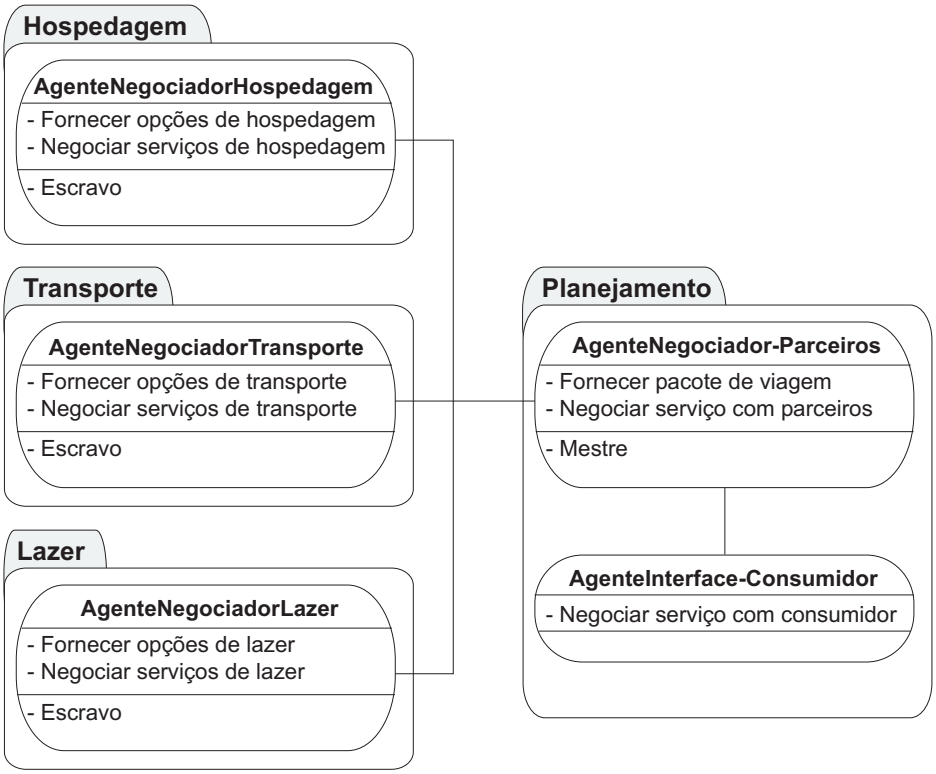

Figura 7.4: Diagrama de agentes para o sistema da empresa *BonVoyage*

Devido à característica de divisão do problema em partes(definição do pacote de viagem) pelo *Agente Negociador-Parceiros* e repassar este problema aos outros agentes negociadores, o protocolo de interação utilizado é o Protocolo Mestre-Escravo [Jun03]. Neste protocolo

definem-se os seguintes papéis de interação: *Mestre*, que representa o particionador e distribuidor do problema, encarregado de reunir os resultados em um resultado final; e *Escravo*, que representa um resolvedor da partição do problema definida pelo mestre.

Os papéis de interação estão representados no diagrama da Figura 7.4, determinando o protocolo de interação entre os agentes. O relacionamento entre os agentes de *Interface-Consumidor* e *Negociador-Parceiros* é apenas uma invocação de serviço, não representando um protocolo de interação específico.

7.6 Definindo os componentes funcionais dos agentes

Após a definição da arquitetura de agentes e dos protocolos que regem a interação entre eles, segundo as diretrizes COMPOR-M, deve-se identificar os componentes que irão implementar as funcionalidades relacionadas às habilidades e papéis de interação destes agentes. O mapeamento das habilidades em serviços dos componentes é descrito a seguir e representado no diagrama de componentes da Figura 7.5.

- **Componente Negociador -** Supondo que as empresas possuam um mesmo mecanismo de negociação, as habilidades *Fornecer opções ** e *Negociar serviço ** são similares nos agentes negociadores, mudando apenas o contexto em que se aplicam. Este componente recebe os parâmetros de negociação e então realiza, de acordo com uma base de regras, esta negociação, fornecendo as opções de pacotes em cada um dos contextos.

- **Componente ACL -** Componente de interação baseada em uma linguagem de comunicação de agentes tal como FIPA-ACL [fIPA02]. Este componente define as mensagens e garante a execução do protocolo Mestre-Escravo. Este componente utiliza o componente de comunicação RMI.

- **Componente Comunicação RMI -** Componente de comunicação para invocação de método remoto entre agentes [Mic03a].

- **Componente Interface Gráfica -** Componente de interface gráfica para a entrada de dados sobre o pacote de viagem por parte do usuário.

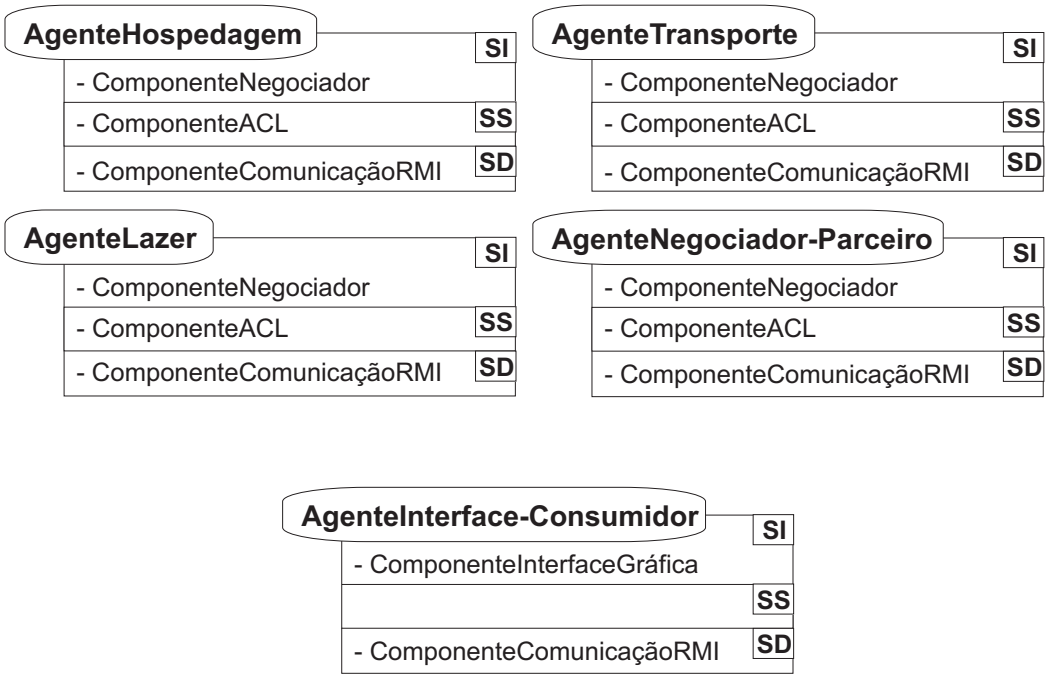

Figura 7.5: Diagrama de componentes dos agentes *BonVoyage*

7.7 "Montando" os agentes com COMPOR-E

Considerando a existência dos componentes descritos anteriormente na paleta de componentes descrita no Capítulo 6, a composição dos agentes é feita através da árvore de agentes, arrastando os componentes da paleta para o agente correspondente na árvore, de acordo com o diagrama de componentes da Figura 7.5. Na Figura 7.6 é apresentada a ferramenta de construção de agentes referente à sociedade de agentes BonVoyage, com os componentes atualmente disponibilizados na paleta de componentes.

Após a montagem da sociedade de agentes, pode-se utilizar a ferramenta de testes para verificar problemas de dependência entre serviços de componentes e, através da ferramenta de monitoramento, monitorar a execução dos agentes da sociedade.

7.8 Flexibilidade da solução diante de cenários evolutivos

No Capítulo 2, são definidas características a serem contempladas pela Engenharia de Software para Sistemas Multiagentes, com base no princípio de dinamicidade e adaptabilidade da solução. A seguir, estas características são relacionadas a cenários evolutivos da empresa

Figura 7.6: Árvore de agentes da sociedade *BonVoyage*

BonVoyage, destacando-se o suporte dado pela infra-estrutura COMPOR para lidar com esta evolução.

1. Flexibilidade para adicionar, remover e "trocar" agentes do sistema, possivelmente em tempo de execução.

   (a) **Cenário evolutivo -** Quando novas empresas se tornarem parceiras de negociação da *BonVoyage*, novos agentes passarão a fazer parte do sistema multiagentes. Da mesma forma, quando as empresas atualmente parceiras deixarem de negociar seus serviços com a *BonVoyage*, agentes deixarão de fazer parte do sistema

multiagentes.

(b) **Suporte da infra-estrutura COMPOR -** Como citado no Capítulo 2, a flexibilidade necessária para este tipo de cenário evolutivo depende da arquitetura definida para o sistema multiagentes. No exemplo da empresa *BonVoyage*, novos agentes podem se anunciar como Escravos ao agente negociador do contexto de planejamento(Mestre), sendo assim, poderiam participar das interações do sistema. O suporte da infra-estrutura COMPOR está relacionado à possibilidade dada ao desenvolvedor de definir qualquer infra-estrutura de comunicação e interação.

2. Flexibilidade para alterar os relacionamentos do sistema multiagentes

   (a) **Cenário evolutivo -** Quando a empresa *BonVoyage* necessitar de um monitoramento dos atributos que influenciam na negociação, por exemplo, para ter um suporte automatizado à decisão da empresa [FCA03], novos agentes de monitoramento serão acoplados ao sistema e deverão se relacionar com os agentes já existentes na sociedade *BonVoyage*.

   (b) **Suporte da infra-estrutura COMPOR -** A flexibilidade necessária para este tipo de cenário evolutivo depende tanto da arquitetura definida para o sistema multiagentes quanto da estrutura de cada um dos agentes. Novos agentes irão interagir, possivelmente através de novos protocolos, por isso, quanto maior a flexibilidade da arquitetura, menos esforço será empregado na adaptação do sistema (Ver item 1(b)). Em relação à estrutura dos agentes, novas habilidades podem surgir ou podem ocorrer alteração nas habilidades existentes para se adequar às características de monitoramento (Ver item 3(b)).

3. Flexibilidade para adicionar, alterar e remover as funcionalidades providas pelos agentes, possivelmente em tempo de execução.

   (a) **Cenário evolutivo -** Quando novos serviços passarem a ser negociados e a forma de negociação se alterar dentro da empresa, pode ser necessário implementar novos algoritmos de negociação. Desta forma, as habilidades dos agentes deverão ser revistas e implementadas novamente.

(b) **Suporte da infra-estrutura COMPOR -** O modelo de componentes COMPOR-CM e o arcabouço COMPOR-F, que são as bases da infra-estrutra de implementação COMPOR, permitem que novos componentes sejam inseridos em tempo de execução nos agentes. Sendo assim, novos serviços podem ser disponibilizados sem que os existentes sejam interrompidos. Esta flexibilidade do projeto de agentes, torna mais simples a adaptação ao cenário evolutivo descrito.

Capítulo 8

Considerações finais

Esta dissertação constituiu-se em uma investigação sobre o tema Engenharia de Software para Sistemas Multiagentes, tendo como ponto de partida resultados obtidos no mencionado ambiente MATHEMA. Nesta perspectiva, foi proposta uma infra-estrutura de engenharia para o desenvolvimento de sistemas de software complexos. Os objetivos descritos foram alcançados tendo como contribuições principais as diretrizes, modelo de componentes, arcabouço e ambiente de desenvolvimento, apresentados nos capítulos anteriores.

Algumas publicações [CAS+02; CASP02; ALF+03; FCA03; CAPP03; APPC03] foram obtidas como forma de validar os resultados gerados, perante a comunidade científica no tema da pesquisa. Sendo assim, cada uma das contribuições foi submetida a congressos nacionais e internacionais com o objetivo de se obter um retorno mais específico e significativo. O trabalho como um todo também foi apresentado em *workshops* de dissertações [1] [ACP03a; ACP03b] a fim de que houvesse retorno em relação à integração das partes, formando uma infra-estrutura para o desenvolvimento de software para sistemas multiagentes.

Por fim, espera-se com este trabalho ter reunido resultados suficientes para a elaboração de um projeto arrojado no tema Engenharia de Software para Sistemas Multiagentes, aplicando as diretrizes, técnicas e ferramentas aqui apresentadas na concepção de aplicações utilizando a abordagem multiagentes e contribuindo para a consolidação desta abordagem na comunidade de Engenharia de Software e Inteligência Artificial.

[1] Esta dissertação foi apresentada como trabalho convidado no WTES/SBES'03 posicionando-se entre os três melhores trabalhos do evento

8.1 Contribuições

Engenharia de software para sistemas multiagentes

Após uma revisão bibliográfica e análise sobre a área de Engenharia de Software aplicada a sistemas multiagentes, constatou-se uma lacuna em relação às justificativas apresentadas para a utilização da abordagem multiagentes no desenvolvimento de sistemas complexos. Os argumentos apresentados neste trabalho não são, em geral, descritos em artigos de conferências e periódicos da área. Em decorrência desse investimento, um artigo de posicionamento no tema foi elaborado [APCP04].

Diretrizes metodológicas

As diretrizes metodológicas apresentadas têm como foco a produção do software, sendo desde o princípio voltada para a especificação de serviços através da abstração de habilidades, não perdendo o vínculo da engenharia do conhecimento através das ontologias. Estas diretrizes representam uma contribuição e um diferencial em relação às demais, devido ao fato de contemplar todas as fases clássicas de desenvolvimento de software. Os resultados referentes às diretrizes metodológicas foram publicados [APPC03].

Modelo de componentes

O modelo de componentes COMPOR-CM possui uma especificação baseada na flexibilidade das entidades que o compõem, possibilitando uma implementação com mudanças de componentes, inclusive em tempo de execução. Este modelo pode ser usado para a construção de outras aplicações onde a flexibilidade seja uma requisito importante. Os resultados referentes ao modelo de componentes foram publicados [CAPP03].

Arcabouço baseado em componentes

O arcabouço baseado em componentes provê a flexibilidade necessária para implementar as especificações do modelo de componentes e as diretrizes metodológicas. Além disso, seu modelo de classes baseado em padrões de projeto é simples e de fácil implementação

em outras linguagens de programação. Os resultados referentes ao arcabouço baseado em componentes foram publicados [CAPP03].

Ambiente de desenvolvimento

O ambiente COMPOR-E para o desenvolvimento de aplicações multiagentes provê ferramentas para auxiliar na construção e execução destas aplicações, dando suporte a todas as fases de desenvolvimento. Os resultados referentes ao ambiente COMPOR-E foram publicados [ALF+03].

Referência para análise bibliográfica

Por fim, este trabalho representa uma boa referência de pesquisa na área de sistemas multiagentes. Os Apêndices B, C e D descrevem, de uma forma não superficial, o "estado da arte" em relação a tecnologias para sistemas multiagentes, servindo como base para futuros trabalhos na área.

8.2 Perspectivas futuras

Diretrizes metodológicas COMPOR-M: aplicações e formalização

- **Utilização das diretrizes para análise e projeto de aplicações multiagentes em domínios diversos**

  A aplicação das diretrizes metodológicas definidas neste trabalho para a análise e projeto de sistemas multiagentes em domínios diversos é indispensável para a consolidação destas diretrizes, verificando sua abrangência e contribuindo também para uma maior validação da abordagem multiagentes como ferramenta para a concepção de software.

- **Definição de uma linguagem para a especificação formal do problema**

  Embora tenham sido definidas diretrizes metodológicas para a especificação do problema e identificação das habilidades, não há uma base formal para garantir que o sistema decorrente de uma determinada especificação estará de acordo com a mesma.

A definição de uma linguagem formal para descrição e posterior validação da especificação de um problema é uma das principais perspectivas futuras deste trabalho.

- **Definição de diretrizes para a especificação de regras sobre as organizações**

  A definição de regras para a interação entre agentes em uma determinada organização tem sido apontada como uma característica importante em sistemas multiagentes [MM03; AMNU00]. É necessário incluir diretrizes para a especificação destas regras. Estas diretrizes podem ser baseadas no arcabouço conceitual FROG [PALA04].

Modelo de componentes COMPOR-CM: verificação formal e análise de desempenho

- **Especificação e análise formal do modelo de componentes COMPOR-CM**

  A especificação formal do modelo de componentes COMPOR-CM possibilitará uma validação de seu funcionamento e análise de propriedades de acordo com determinadas configurações arquiteturais. Sendo o modelo a base para toda a estrutura de implementação e execução deste trabalho, uma formalização do modelo tem uma importância considerável. Uma vez que já existem alguns resultados em busca desta especificação [SAPC03], esta perspectiva tende a se tornar realidade em um futuro próximo.

- **Análise de desempenho do modelo de componentes COMPOR-CM**

  A flexibilidade do modelo de componentes apresentada neste trabalho pode ser dispendiosa em termos de processamento dependendo da quantidade de componentes que fazem parte de um determinado sistema. Apesar de considerar a perda pouco significante, uma análise de desempenho do modelo em relação a outros modelos correlatos é importante e representa um relevante e provável futuro investimento.

Arcabouço COMPOR-F: heterogeneidade de linguagem

- **Implementação do arcabouço em outras linguagens de programação**

  Uma vez que o arcabouço aqui apresentado foi implementado em Java, um possível investimento futuro é a implementação do arcabouço em outras linguagens orientadas

a objetos, como por exemplo C++. Sendo a comunicação entre os agentes realizada via rede, este seria o ponto de interoperabilidade entre os agentes Java e os agentes C++, por exemplo. Para tanto, tecnologias como CORBA [Gro03] podem ser utilizadas. Sendo o projeto do arcabouço de simples codificação, a dificuldade ficaria por conta da sua utilização, a qual, neste trabalho, possui o indispensável auxílio do ambiente COMPOR-E.

Ambiente COMPOR-E: acoplamento de novas ferramentas

- **Disponibilização de modelos arquiteturais**

  Apesar da disponibilização de componentes auxiliar na construção de aplicações multiagentes permitindo que as funcionalidades dos agentes sejam acopladas via *drag and drop*, o ambiente não provê suporte para arquiteturas com conjuntos de componentes pré-definidos. Por exemplo, uma arquitetura de agentes baseada em *blackboard* [SG96] teria uma estrutura de memória compartilhada e agentes capazes de escrever e ler informações nesta estrutura. Sendo assim, os agentes desta arquitetura, independente do domínio de aplicação, teriam, na maioria dos casos, os mesmos componentes de interação. Por isso, a disponibilização de modelos arquiteturais pré-definidos no ambiente COMPOR-E é de extrema importância para o aumento da produtividade em relação à construção de sociedades de agentes.

- **Automatização do processo de mapeamento de habilidades em componentes**

  Uma das perspectivas futuras descritas anteriormente é a definição de uma linguagem formal para a especificação do problema e das habilidades dos agentes. Uma vez cumprida esta etapa, seria possível um mapeamento das habilidades dos agentes para os serviços providos pelos componentes. Desta forma o processo de construção dos agentes, após a identificação das habilidades, poderia ser automatizado. A abordagem para a recuperação de componentes baseada em modelos definida em [SP03] poderia ser adaptada e utilizada.

- **Construção e disponibilização de componentes funcionais**

  A biblioteca de componentes funcionais disponibilizada pelo ambiente não possui

componentes suficientes para a utilização do ambiente em larga escala. Componentes de acesso a Ontologias [Pro03], inferência baseada em regras de produção e *frames* [FH02], comunicação via SOAP [BEK+00], entre outros, devem ser desenvolvidos para disponibilizar um maior acervo de serviços para o desenvolvedor.

- **Inserção de um módulo de software que garanta o cumprimento das leis sobre as organizações**

  Uma vez definidas as diretrizes para a especificação de leis sobre as organizações de agentes, é necessário que haja um arcabouço de software que garanta o cumprimento destas leis a nível de implementação. O arcabouço JFROG [PALA04] pode ser utilizado para acoplar esta funcionalidade ao ambiente COMPOR-E.

Estudo de casos: desenvolvimento de novas aplicações

- **Desenvolvimento de aplicações utilizando as ferramentas apresentadas**

  O desenvolvimento de aplicações multiagentes nos domínios de Recuperação de Informação, Sistemas Tutores Inteligentes, Redes de Sensores, entre outros, utilizando as ferramentas de Engenharia de Software propostas neste trabalho, tendem a contribuir com a consolidação, evolução e inevitáveis retificações das mesmas.

- **Analisar empiricamente o custo/benefício do desenvolvimento de sistemas utilizando uma abordagem multiagentes**

  Com base nas experiências obtidas no desenvolvimento das aplicações multiagentes, identificar as vantagens e desvantagens da utilização da abordagem multiagentes para o desenvolvimento de software em relação a outras abordagens.

Bibliografia

[AA00] Henri Avancini e Analía Amandi. The Open Agent Architecture: a framework for building distributed software systems. *SADIO Electronic Journal of Informatics and Operations Research*, 3(1):1–12, 2000.

[ACP03a] Hyggo Oliveira Almeida, Evandro Barros Costa e Angelo Perkusich. COMPOR: uma metodologia para o desenvolvimento de sistemas multi-agentes. *Workshop de Dissertações da COPIN*, Campina Grande, Brasil, Agosto 2003.

[ACP03b] Hyggo Oliveira Almeida, Evandro Barros Costa e Angelo Perkusich. COMPOR: uma metodologia para o desenvolvimento de sistemas multi-agentes. *Workshop de Teses em Engenharia de Software, Simpósio Brasileiro de Engenharia de Software - SBES'03*, volume 8, págs. 95–100, Manaus, Brasil, Outubro 2003.

[ALF+03] Hyggo Oliveira Almeida, Emerson Loureiro, Glauber Ventura Ferreira, Rodrigo Barros Paes, Angelo Perkusich e Evandro Barros Costa. Ambiente integrado para o desenvolvimento de sistemas multi-agentes. *Proceedings of Simpósio Brasileiro de Engenharia de Software - Sessão de ferramentas - SBES 2003*, volume 17, págs. 55–60, Manaus, Brasil, 2003.

[AMNU00] Xuhui Ao, Naftaly H. Minsky, Thu D. Nguyen e Victoria Ungureanu. Law-Governed Internet Communities. *4th International Conference Coordination Models and Languages - COORDINATION 2000*, volume 1906 of *Lecture Notes in Computer Science*, págs. 133–147. Springer Verlag, 2000.

[AP97] Analía Amandi e A. Price. Object-oriented agent programming through the

brainstorm system. *PAAM'97 (Practical Applications of Intelligent Agents and Multi-Agents), London*, Abril 1997.

[APCP04] Hyggo Oliveira Almeida, Rodrigo Barros Paes, Evandro Barros Costa e Angelo Perkusich. Why Are Agents Good for Engineering Multiagent Systems? Submetido a Third International Joint Conference on Autonomous Agents and Multiagent Systems - AAMAS'04, Nova Iork, EUA, 2004.

[APPC03] Hyggo Oliveira Almeida, Angelo Perkusich, Rodrigo Barros Paes e Evandro Barros Costa. COMPOR: uma metodologia para o desenvolvimento de sistemas multi-agentes. *Proceedings of III Jornada Ibero-americana de Engenharia de Software e Engenharia do Conhecimento*, Valvidia, Chile, Novembro 2003.

[AZI99] Analía Amandi, Alejandro Zunino e Ramiro Iturregui. Multi-paradigm Languages Supporting Multi-agent Development. Francisco J. Garijo e Magnus Boman, editores, *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99)*, volume 1647, págs. 128–139. Springer-Verlag: Heidelberg, Alemanha, 30– 2 1999.

[BEK+00] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte e Dave Winer. Simple Object Access Protocol - W3C Note 08. Technical report, W3C, Maio 2000.

[BFP00] Calebe Paula Bianchini, Diogo Sobral Fontes e Antonio Francisco Prado. A Distributed Software Agents Platform Framework. *1st International Workshop on Software Engineering for Large-Scale Multi-Agent Systems - SELMAS'2002 - Orlando/Flórida - EUA*, 2000.

[BG98] Kent Beck e Erich Gamma. JUnit Test Infected: Programmers Love Writing Tests - Java Report. http://junit.sourceforge.net/doc/testinfected/testing.htm - Acessado em 01/02/2004, 1998.

[BRHL99] Paolo Busetta, Ralph Rönnquist, Andrew Hodgson e Andrew Lucas. JACK Intelligent Agents - Components for Intelligent Agents in Java. AgentLink News Letter. White paper, http://www.agent-software.com.au. - Acessado em 01/02/2004, Janeiro 1999.

[BRJ00] Grady Booch, James Rumbaugh e Ivar Jacobson. *UML - Guia do usuário*. Campus, 2000.

[BRP99] F. Bellifemine, G. Rimassa e A. Poggi. JADE - A FIPA-Compliant Agent Framework. *Proceedings of the 4th International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, Inglaterra*, págs. 97–108, 1999.

[Bur96] B. Burmeister. Models and Methodology for Agent-Oriented Analysis and Design. *Working Notes of the KI'96 Workshop on Agent-Oriented Programming and Distributed Systems*. DFKI Document, Saarbrücken, Alemanha, 1996.

[CAL+01] E.B. Costa, H. O. Almeida, E. F. A. Lima, R. R. G. Nunes Filho, K. S. Silva, e F. M. Assunção. SHART-Web - Um Sistema Tutor Multiagentes em Harmonia na Web. *Proceedings of III Workshop Ambientes de aprendizagem baseados em agentes - SBIE2001 - Vitória - ES*, Novembro 2001.

[CAL+02] E.B. Costa, H. O. Almeida, E. F. A. Lima, R. R. G. Nunes Filho, K. S. Silva, e F. M. Assunção. A Cooperative Intelligent Tutoring System: The case of Musical Harmony domain. C. Coello, A. Albornoz, L. Sucar e O Battistuti, editores, *Proceedings of 2nd Mexican International Conference on Artificial Intelligence - MICAI'02*, volume 2313 of *Lecture Notes in Artificial Intelligence*, págs. 367–376, Mérida, Yucatán, México, Abril 2002. Springer Verlag.

[CAPP03] Evandro Barros Costa, Hyggo Oliveira Almeida, Rodrigo Barros Paes e Angelo Perkusich. COMPOR: a Component-based Framework for building Multiagent Systems. *Proceedings of 2nd International Workshop on Software Engineering for Large-Scale Multi-agent Systems - on ICSE 2003 - Portland - EUA*, págs. 84–89, Maio 2003.

[CAS+02] Evandro Barros Costa, Hyggo Oliveira Almeida, Klebson Santos Silva, Angelo Perkusich e Rodrigo de Barros Paes. Towards a Methodology for Building Intelligent Tutoring Systems in a Distance Learning Environment Based on a Multi-agent Approach. *Proceedings of IV Workshop Ambientes de aprendizagem baseados em agentes - SBIE2002 -Cidade Universitaria - RS*, págs. 617–620, Novembro 2002.

[CASP02] Evandro Barros Costa, Hyggo Oliveira Almeida, Klebson Santos Silva e Angelo Perkusich. A Conceptual Framework for Building Intelligent Tutoring Systems in a Distance Learning Environment Based on a Multi-agent Approach. *Proceedings of Workshop Architectures and Methodologies for Building Agent-based learning Environments - ITS 2002 - San Sebastian - Espanha*, págs. 48–57, Junho 2002.

[CCG+01] Giovanni Caire, Wim Coulier, Francisco J. Garijo, Jorge Gomez, Juan Pavon, Francisco Leal, Paulo Chainho, Paul E. Kearney, Jamie Stark, Richard Evans, e Philippe Massonet. Agent Oriented Analysis Using Message/UML. *AOSE*, págs. 119–135, 2001.

[CFA02] E.B. Costa, E. Ferneda e H. O. Almeida. Um Sistema Tutor Inteligente para o ensino da Harmonia Musical: uma abordagem multi-agentes. *Proceedings of IV Workshop Ambientes de aprendizagem baseados em agentes - SBIE2002 -Cidade Universitaria - RS*, págs. 621–624, Novembro 2002.

[CGAP02] R. Courdier, F. Guerrin, F.H. Andriamasinoro e J.M. Paillat. Agent-based simulation of complex systems: Application to collective management of animal wastes. *Journal of Artificial Societies and Social Simulation (JASSS)*, 5(3), Junho 2002.

[CGFP98] E. B. Costa, G. M. Góis, J. C. A. Figueiredo e A. Perkusich. Towards a multi-agent interactive learning environment oriented to the petri net domain. *Proc.of IEEE Int. Conf. on Systems Man and Cybernetics*, págs. 250–261, San Diego, EUA, Outubro 1998. Springer-Verlag.

[Cha97] Deepika Chauhan. *JAFMAS: A Java-based Agent Framework for Multiagent Systems Development and Implementation*. PhD thesis, ECECS Department Thesis, University of Cincinnati, Cincinnati, OH, 1997.

[CKM01] J. Castro, M. Kolp e J. Mylopoulos. A requirements-driven development methodology. *13th Int. Conf. on Advanced Information Systems Engineering - CAiSE'01*, págs. 108–123, Interlaken, Suíça, Junho 2001.

[CLF95] E. B. Costa, M. A. Lopes e E. Ferneda. MATHEMA: A Learning Environment Based on a Multi-Agent Architecture. J. Wainer e A. Carvalho, editores, *Proceedings of the 12th Brazilian Symposium on Artificial Intelligence*, volume 991 of *Lecture Notes in Artificial Intelligence*, págs. 141–150, Campinas, Brasil, 1995. Springer-Verlag.

[CMB00] D. Camacho, J. Molina e D. Borrajo. A multiagent approach for electronic travel planning. *Proceedings of the Second International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2000)*, Austin, EUA, julho 2000. AAAI Press.

[Cos97] Evandro Barros Costa. *Um Modelo de Ambiente Interativo de Ensino e Aprendizagem Baseado numa Arquitetura Multi-Agentes*. Tese de doutorado, Departamento de Engenharia Elétrica - Universidade Federal da Paraíba, Outubro 1997.

[Cos99] Evandro Barros Costa. Projeto de Pesquisa em Informática na Educação MATH-NET, Uma abordagem via sistemas multiagentes para concepção e realização de ambientes interativos de aprendizagem cooperativa assistidos por computador, ProTeM-CC - PTI - PEDU, 1999.

[CP99] M. Campo e T. Price. Luthier - a framework for building program analysis tools. *Implementing Applications Frameworks: Object Oriented Frameworks*. Wiley, 1999.

[CPF98] Evandro Barros Costa, Angelo Perkusich e Edilson Ferneda. From a Tridimensional view of Domain Knowledge to Multi-agent Tutoring System. *Proce-*

edings of 14th Brazilian Symposium on Artificial Intelligence - Lecture notes on artificial intelligence, LNAI 1515, volume 991, págs. 61–72, Porto Alegre, Brasil, Novembro 1998. Springer Verlag.

[Crn01] Ivica Crnkovic. Component-based Software Engineering - New Challenges in Software Development. *Software Focus*, volume 4, págs. 127–133. Wiley, Dezembro 2001.

[DAM00] DAML. The DARPA Agent Markup Language Homepage. http://www.daml.org/ - Acessado em 01/02/2004, Agosto 2000.

[DeL02] Scott A. DeLoach. Modeling Organizational Rules in the Multiagent Systems Engineering Methodology. *15th Canadian Conference on Artificial Intelligence (AI'2002)*, Calgary, Alberta, Canada, Maio 2002.

[DFH96] John Davies, Dieter Fensel e Frank Van Harmelen. *Towards the Semantic Web - Ontology-driven Knowledge Management*. Prentice Hall, 1996.

[DW03] D. D'Souza e A. Wills. Catalysis - 1997 - http://www.catalysis.org/ - Acessado em 01/02/2004, 2003.

[DWS01] S. Deloach, M. Wood e C. H. Sparkman. Multiagent Systemas Engineering. *International Journal of Software Engineering and Knowledge Engneering*, volume 11:3, págs. 231–258. World Scientific Publishing Company, 2001.

[DZ98] A. Drogoul e J. D. Zucker. Methodological issues for designing multi-agent systems with machine learning techniques: capitalizing experiences from the RoboCup Challenge. Technical report, LIPS Research Reports, Outubro 1998.

[Ecl03] Eclipse. Eclipse Project. http://www.eclipse.org - Acessado em 01/02/2004, 2003.

[Emo03] Emorphia. FIPA-OS. http://fipa-os.sourceforge.net - Acessado em 01/02/2004, 2003.

[FCA03] Ricardo Rubens Gomes Nunes Filho, Evandro Barros Costa e Hyggo Oliveira Almeida. An Architecture and A Decision-Making Model to Web-Based Electronic Commerce. *Proceedings of of I3E - 3rd IFIP conference on e-Commerce, e-Business and e-Government - São Paulo - Brasil*, págs. 117–128, 2003.

[Fer99] Jacques Ferber. *Multi-Agent System: An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.

[FFMM94] T. Finin, R. Fritzson, D. McKay e R. McEntire. KQML as an Agent Communication Language. N. Adam, B. Bhargava e Y. Yesha, editores, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, págs. 456–463, Gaithersburg, EUA, 1994. ACM Press.

[FG02] J. H. A. Silva Filho e Rosario Girardi. SIMCAP: Um Sistema Multiagent para a captura de publicações científicas na web. *Revista Eletrônica de Iniciação Científica da Sociedade Brasileira de Computação*, 2(1), 2002.

[FH02] Ernest Friedman-Hill. JESS - The rule engine for the Java Platform. http://herzberg.ca.sandia.gov/jess/ - Acessado em 01/02/2004, 2002.

[fIPA02] Foundation for Intelligent Physical Agents. FIPA ACL Message Structure Specification. http://www.fipa.org/specs/fipa00061/ - Acessado em 01/02/2004, Dezembro 2002.

[FJS00] Mohamed Fayad, Ralph Johnson e Douglas Schmidt. *Building application frameworks*. Wiley, 2000.

[GD99] J. Graham e K. Decker. Towards a Distributed, Environment-Centered Agent Framework. *Proc. of International Workshop on Agent Theories, Architectures and Languages (ATAL99)*, págs. 290–304, 1999.

[Gen98] Michael R. Genesereth. Knowledge Interchange Format. Draft proposed American National Standard - NCITS.T2/98-004, 1998.

[GFM03] Olivier Gutknecht, Jacques Ferber e Fabien Michel. Multi-Agent Development Kit. http://www.madkit.org/ - Acessado em 01/02/2004, 2003.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides. *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, 1995.

[Gir01] Maria Rosario Girardi. Um ambiente para engenharia de aplicações multiagentes (MaAE), Programa Temático e Multi-institucional em Ciência da Computação (ProTeM-CC), 2001.

[GL02] Alessandro Garcia e Carlos Lucena. Software Engineering for Large-Scale Multi-Agent Systems, SELMAS 2002, Post-Workshop Report. Technical report, ACM Software Engineering Notes, Agosto 2002.

[Gla97] N. Glasser. The CoMoMAS Approach: From Conceptual Models to Executable Code. *Proc. of 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW-97)*, Ronneby, Suécia, 1997.

[GMP02] Fausto Giunchiglia, John Mylopoulos e Anna Perini. The tropos software development methodology: processes, models and diagrams. *1st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 02)*, págs. 35–36, Bologna, Itália, 2002.

[Gra99] Mark Grand. *Patterns in Java*, volume 1. Wiley, 1999.

[Gre03] Dale Green. The Java Tutorial - Trail: Internationalization. http://java.sun.com/ - Acessado em 01/02/2004, 2003.

[Gri94] Martin L. Griss. Software Reuse: Objects and Frameworks are not Enough. Technical Report HPL-95-03, Hewlett-Packard Laboratories, Janeiro 1994.

[Gro03] Object Management Group. CORBA FAQ and Resources. http://www.omg.org - Acessado em 01/02/2004, 2003.

[GSP98] Steven Goldsmith, Shannon Spires e Laurence Phillips. Object Frameworks for Agent System Development. *Software Tools for Developing Agents - AAAI Workshop*, págs. 107–112, Julho 1998.

[HJF95] J. Huang, N. R. Jennings e J. Fox. An agent-based approach to health care management. *Applied Artificial Intelligence: An International Journal*, 9(4):401–420, 1995.

[HRHL01] N. Howden, R. Ronnquist, A. Hodgson e A. Lucas. JACK Intelligent Agents - Summary of an Agent Infrastructure. *Proceedings of the 5th International Conference on Autonomous Agents*, Montreal, Canada, 2001.

[HSAG03a] Z.I. Hashmi, C. Yu-N S.S.R. Abidi e L.K. Guan. A Generic Intelligent Agent Design: Towards A Multiagent Healthcare Knowledge Management System. *IADIS International Conference WWW/Internet 2003*, Algarve, Portugal, Novembro 2003.

[HSAG03b] Z.I. Hashmi, C. Yu-N S.S.R. Abidi e L.K. Guan. Adaptive User Interface for Multimedia Communication System Based on Multiagent . *17 th International Conference on Advanced Information Networking and Applications (AINA'03)*, págs. 53–58, Xian, China, Março 2003. IEEE Press.

[IGG98] C. A. Iglesias, M. Garijo e J. C. Gonzales. Analysis and Design of Multiagent Systems Using MAS-CommonKADS. M.P. Singh, A.S. Rao e M.J. Wooldridge, editores, *Intelligent Agents IV (ATAL'97)*, volume 1365 of *Lecture Notes in Artificial Inteligence*, págs. 314–327, Berlim, Alemanha, 1998. Springer Verlag.

[IGG99] Carlos Iglesias, Mercedes Garrijo e José Gonzalez. A Survey of Agent-Oriented Methodologies. Jörg Müller, Munindar P. Singh e Anand S. Rao, editores, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures and Languages (ATAL-98)*, volume 1555, págs. 317–330. Springer-Verlag: Heidelberg, Alemanha, 1999.

[Jen99] Nicholas R. Jennings. Agent-Oriented Software Engineering. Francisco J. Garijo e Magnus Boman, editores, *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World : Multi-Agent System Engineering (MAAMAW-99)*, volume 1647, págs. 1–7. Springer-Verlag: Heidelberg, Alemanha, 30– 2 1999.

[Jen01] Nicholas R. Jennings. An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41, 2001.

[Joe02] Joel Moses. The Anatomy of Large Scale Systems. *Working Paper Series - MIT Engineering Systems Division*. MIT Press, 2002.

[Jos02] Joseph M. Sussman. Collected Views on Complexity in Systems. *Working Paper Series - MIT Engineering Systems Division*. MIT Press, 2002.

[JPC00] Heecheol Jeon, Charles Petrie e Mark R. Cutkosky. JATLite: A Java Agent Infrastructure with Message Routing. *IEEE Internet Computing*, 4(2):87–96, Julho 2000.

[Jun03] Geovane Bezerra Silva Junior. Padrões Arquiteturais para o Desenvolvimento de Aplicações Multiagente. Master's thesis, Universidade Federal do Maranhão, 2003.

[KGR96] David Kinny, Michael Georgeff e Anand Rao. A Methodology and Modelling Technique for Systems of BDI Agents. Rudy van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, Holanda, 1996.

[KKPC00] E. A. Kendall, P.V. M. Krishna, C. V. Pathak e C.B.Suresh. A Java Application Framework for Agent Based Systems. ACM Computing Surveys Symposium on Application Frameworks, 2000.

[KLM02] M. Klusch, S. Lodi e G.L. Moro. Agent-Based Distributed Data Mining: The KDEC Scheme. *Intelligent Information Agents: The AgentLink Perspective*, volume 2586 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2002.

[Klu00] M. Klusch. *Agent Technology applied to Networked Systems*, chapter 5: Agent-Mediated Trading: Intelligent Agents and E-Business. John Wiley e Sons, 2000.

[KMJ95] E. A. Kendall, M. T. Malkoun e C. H. Jiang. A Methodology for Developing Agent Based Systems. Chengqi Zhang e Dickson Lukose, editores, *First Aus-*

tralian Workshop on Distributed Artificial Intelligence, Canberra, Australia, 1995.

[Lab03] Multiagent & Cooperative Robotics Lab. Agent Tool Project. http://www.cis.ksu.edu/ sdeloach/ai/agentool.htm - Acessado em 01/02/2004, 2003.

[Lar00] Craig Larman. *Utilizando UML e Padrões - Uma introdução à análise e ao projeto orientados a objetos*. Bookman, 2000.

[LFP99] Yannis Labrou, Tim Finin e Yun Peng. Agent Communication Languages: The Current Landscape. *IEEE Intelligent Systems*, 14(2):45–52, 1999.

[Lin00a] J. Lind. A development method for multiagent systems. *Proceedings of the 15th European Meeting on Cybernetics and Systems Research, Symposium From Agent Theory to Agent Implementation*, 2000.

[Lin00b] J. Lind. *MASSIVE: Software Engineering for Multiagent Systems*. PhD thesis, University of the Saarland, Alemanha, 2000.

[Lin00c] Jurgen Lind. The Massive Development Method for Multiagent Systems. Jeffrey Bradshaw e Geoff Arnold, editores, *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)*, págs. 339–354, Manchester, Inglaterra, 2000. The Practical Application Company Ltd.

[LOT03] Victor Lesser, Charles L. Ortiz e Milind Tambe, editores. *Distributed Sensor Networks: A Multiagent Perspective*. Kluwer Academic Publishers, 2003.

[LS95] R. Greg Lavender e Douglas C. Schmidt. Active Object: an Object Behavioral Pattern for Concurrent Programming. *Proc.Pattern Languages of Programs,*, 1995.

[Lud76] Ludwig Von Bertalanffy. *General System Theory: Foundations, Development, Applications (Revised Edition)*. George Braziller, Março 1976.

[LW01] Jiming Liu e Jianbing Wu. *Multiagent Robotic Systems*. CRC Press, Maio 2001.

[Mae87] P. Maes. Concepts and experiments in computational reflection. *ACM SIGPLAN Notices*, 22(12): 147– 155, Dezembro 1987.

[MCM99] D. Martin, A. Cheyer e D. Moran. The Open Agent Architecture: a framework for building distributed software systems. *Applied Artificial Intelligence*, 13(1/2):91–128, 1999.

[Mic97] Sun Microsystems. Java Beans Specification. http://java.sun.com/products/javabeans/docs/spec.html - Acessado em 01/02/2004, 1997.

[Mic03a] Sun Microsystems. Remote Method Invocation. http://java.sun.com - Acessado em 01/02/2004, 2003.

[Mic03b] Sun Microsystems. The Reflection API. http://java.sun.com/docs/books/tutorial/reflect/ - Acessado em 01/02/2004, 2003.

[MKC01] John Mylopoulos, Manuel Kolp e Jaelson Castro. UML for Agent-Oriented Software Development: The Tropos Proposal. *Lecture Notes in Computer Science*, 2185:422–441, 2001.

[MM03] Takahiro Murata e Naftaly H. Minsky. On Monitoring and Steering in Large-Scale Multi-Agent Systems. *Selmas'03 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*, Portland, Oregon, EUA, Maio 2003.

[Omi00] A. Omicini. SODA: Societies and Infrastructures in the Analysis and Design of Agent- Based Systems. *Agent Oriented Software Engineering*, Limerick, Irlanda, 2000.

[ON98] P.D. O'Brien e R.C. Nicol. FIPA - Towards a standard for software agents. *BT Technology Journal*, 16(3):51–59, Julho 1998.

[ON00] Kok-Leong Ong e Wee-Keong Ng. A Survey of Multi-Agent Interaction Techniques and Protocols. Technical Report CAIS-TR04-98, School of Applied Science, Technological University, Nanyang, Singapura, 2000.

[OPB00] James Odell, H. Van Dyke Parunak e Bernhard Bauer. Representing Agent Interaction Protocols in UML. *Agent Oriented Software Engineering*, págs. 121–140, 2000.

[Pag03] The CORBA Component Model (CCM) Page. CORBA Component Model Specification. http://ditec.um.es/ dsevilla/ccm/ - Acessado em 01/02/2004, 2003.

[PALA04] Rodrigo Barros Paes, Hyggo Oliveira Almeida, Carlos Lucena e Paulo Alencar. Enforcing Secure Interaction Protocols in Multi-Agent Systems. Submetido a Third International Joint Conference on Autonomous Agents and Multiagent Systems - AAMAS'04, Nova Iork, EUA, 2004.

[Pre95] Roger Pressman. *Engenharia de Software*. MacronBooks, 1995.

[Pro03] Protege. Protege ontology and knowledge-base editor. http://protege.stanford.edu/ - Acessado em 01/02/2004, 2003.

[PSS02] T.R. Payne, R. Singh e K. Sycara. RCAL: A Case Study on Semantic Web Agents. *1st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 02)*, Nova Iork, EUA, 2002. ACM Press.

[PW02] Lin Padgham e Michael Winikoff. Prometheus: A pragmatic methodology for engineering intelligent agents. *Proceedings of Workshop on Agent Oriented Methodologies, OOPSLA 2002*, Seattle, EUA, 2002.

[RAJ01] Ed Roman, Scott W. Ambler e Tyler Jewell. *Mastering Enterprise JavaBeans*. John Wiley and Sons, 2001.

[RDF97] RDF. Resource Description Framework. http://www.w3.org/RDF/ - Acessado em 01/02/2004, 1997.

[Ret03] Retsina. Retsina agent architecture. http://www-2.cs.cmu.edu/ softagents/ - Acessado em 01/02/2004, 2003.

[RG95] A. S. Rao e M. P. Georgeff. BDI-agents: from theory to practice. *Proceedings of the First Intl. Conference on Multiagent Systems*, págs. 312–319, San Francisco, EUA, Junho 1995.

[RPS02] Thomas Ruan, Adrian Pearce e Leon Sterling. ROADMAP: Extending Gaia Methodology for Complex Open Systems. *1st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 02)*, Bologna, Itália, 2002.

[SAPC03] Leandro Dias Silva, Hyggo Oliveira Almeida, Angelo Perkusich e Evandro Barros Costa. Modelling and Analysis of a Multi-Agent Intellgent Tutoring System Based on Coloured Petri Nets. *1st ACIS International Conference on Software Engineering Research and Applications (SERA'03)*, volume 1, págs. 276–281, San Francisco, EUA, 2003. Mt. Pleasant: International Association for Computer and Information Sciences (ACIS).

[Sel99] John Self. The defining characteristics of intelligent tutoring systems research: ITSs care, precisely. *International Journal of Artificial Intelligence in Education*, 10:350–364, 1999.

[SG96] M. Shaw e D. Garlan. *Software Architecture: Perspective on an emerging discipline*. Prentice Hall, 1996.

[Sho93] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, number 60 in 1, págs. 51–92. Elseiver, 1993.

[Sod01] Alidia Glícia Silva Sodré. Metodologia baseada em agentes para o desenvolvimento de software - MADS. Master's thesis, Universidade Federal do Maranhão, São Luís, Brasil, 2001.

[SP03] Leandro Dias Silva e Angelo Perkusich. Formal Verification of Component-Based Software Systems. *Proceedings of The First International Workshop on Verification and Validation of Enterprise Information Systems - WEIS'03*, Angers, França, 2003.

[SWH+94] A.T. Schreiber, B.J. Wielinga, R. Hoog, J.M. Akkermans e W.V. Velde. CommonKADS: A comprehensive methodology for KBS development. *IEEE Expert*, volume 9(6), págs. 28–37. IEEE Computer Society, Dezembro 1994.

[Szy98] Clemens Szypersky. *Component Software, Beyond Object-Oriented Programming*. Addison-Wesley, 1998.

[TEC03] TECCOM. Projeto ESSMA - Engenharia de Software de Sistemas Multi-Agentes. http://www.teccomm.les.inf.puc-rio.br/essma/ - Acessado em 01/02/2004, 2003.

[Ver96] François B. Vernadat. *Enterprise Modeling and Integration: principles and applications*. Chapman and Hall, 1996.

[VHL01] Regis Vincent, Bryan Horling e Victor Lesser. An Agent Infrastructure to Build and Evaluate Multi-agent Systems: The Java Agent Framework and Multi-agent System Simulator. *Lecture Notes in Computer Science*, 1887:102–127, 2001.

[Wat97] Ian Watson. *Applying Case-Based Reasoning : Techniques for Enterprise Systems*. Morgan Kaufmann, Julho 1997.

[WJK00] Michael Wooldridge, Nicholas R. Jennings e David Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.

[Woo98] M. Wooldridge. Agents and software engineering. *AI*IA Notizie*, XI(3):31–37, 1998.

[WW99] William E. Walsh e Michael P. Wellman. Modeling supply chain formation in multiagent systems. *Agent Mediated Electronic Commerce (IJCAI Workshop)*, págs. 94–101, 1999.

[ZA00] Alejandro Zunino e Analía Amandi. Brainstorm/J: a Java Framework for Intelligent Agents. *Proc. of the* 2^{nd} *Argentinian Symposium on Artificial Intelligence (ASAI 2000 -* 29^{th} *JAIIO)*, págs. 43–58, Tandil, Buenos Aires, Argentina, 2000.

[ZAH03] S.Z.H. Zaidi, S.S.R. Abidi e Z.I. Hashmi. Engineering of Multi-Agent Systems to Effectuate Distributed Data Mining Activities. *18th International Congress of the European Federation for Medical Informatics Medical Informatics in Europe*, St. Malo, França, Maio 2003.

[Zeu03] Zeus. Zeus Agent System. http://more.btexact.com/projects/agents/zeus/index.htm - Acessado em 01/02/2004, 2003.

[ZJW03] Franco Zambonelli, Nicholas R. Jennings e Michael Wooldridge. Developing multiagent systems: The Gaia methodology. *ACM Trans. Softw. Eng. Methodol.*, 12(3):317–370, 2003.

[ZP04] F. Zambonelli e V. Parunak. Towards a Paradigm Change in Computer Science and Software Engineering: a Synthesis. *The Knowledge Engineering Review - A ser publicado*, 2004.

Apêndice A

Notação definida para a construção de modelos em COMPOR - COMPOR-L

Neste apêndice é apresentada com detalhes a notação COMPOR-L [1] para a documentação e representação de cada fase de desenvolvimento definida nas diretrizes metodológicas COMPOR-M. Os diagramas aqui apresentados seguem a mesma seqüência definida no Capítulo 3.

Diagrama de Habilidades

O diagrama de habilidades recebe este nome porque tem como foco a representação gráfica das habilidades e suas dependências. Além disso, os contextos a que pertencem também são representados neste diagrama. Os elementos que fazem parte deste diagrama são descritos, separadamente, a seguir. Na Figura A.1 é apresentado o diagrama de habilidades de COMPOR-L.

Contexto

Um Contexto é representado pelo símbolo ilustrado na Figura A.2. Como descrito no Capítulo 3, um contexto representa uma partição do domínio de conhecimento. Sendo assim, um contexto determina a partição do domínio ao qual se referem as suas habilidades.

[1] Abreviação para COMPOR-*Language*

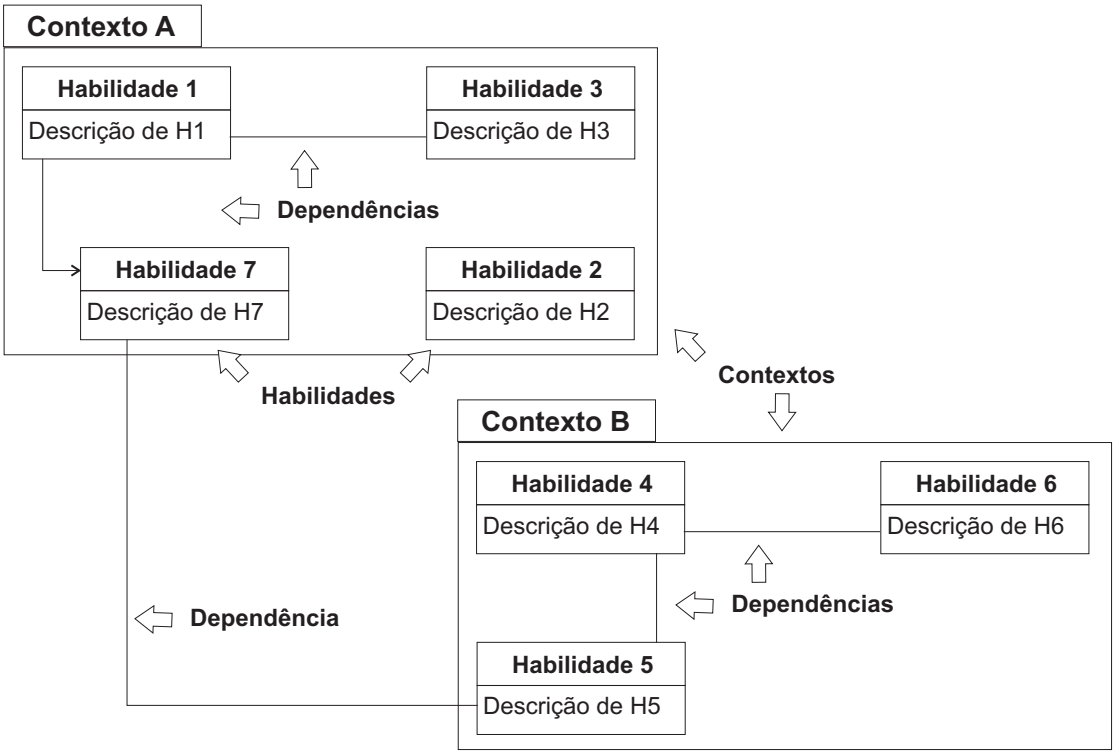

Figura A.1: Elementos do Diagrama de Habilidades

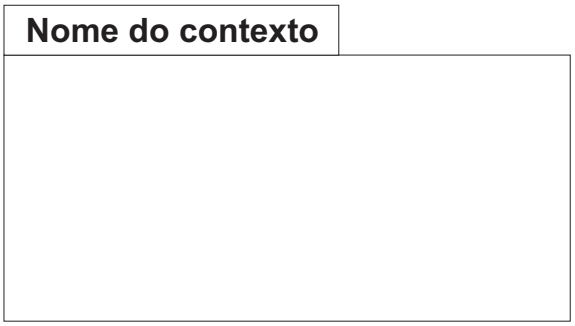

Figura A.2: Representação gráfica de um Contexto

Habilidade

Na Figura A.3 é ilustrada a representação gráfica de uma Habilidade. Para cada habilidade é definido um nome identificador e uma breve descrição. Duas habilidades diferentes não podem ter o mesmo identificador.

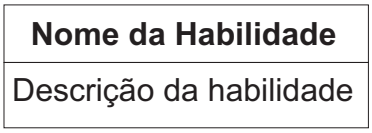

Figura A.3: Representação gráfica de uma Habilidade

Dependência

As dependências entre as habilidades são descritas através de arcos de ligação entre os símbolos representativos das habilidades. O sentido do arco define a dependência da habilidade de origem do arco em relação à habilidade destino e a ausência de sentido indica dependência mútua. Na Figura A.4 são apresentados exemplos de dependências entre habilidades. As dependências podem existir entre habilidades de contextos diferentes.

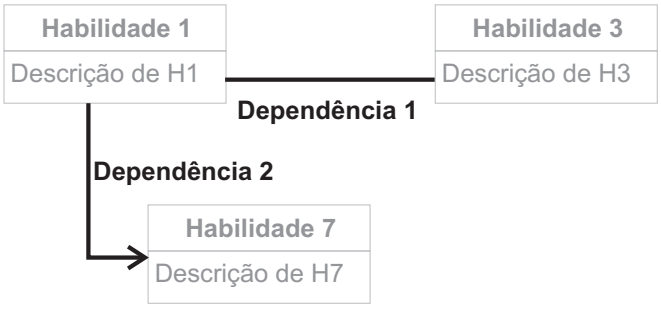

Figura A.4: Representação gráfica de uma Dependência

Diagrama de Agentes

O diagrama de agentes representa graficamente os agentes do sistema e suas associações. Além disso, as organizações a que pertencem também são representadas neste diagrama. Os elementos que fazem parte deste diagrama são descritos, separadamente, a seguir. Na Figura A.5 é apresentado o diagrama de agentes de COMPOR-L.

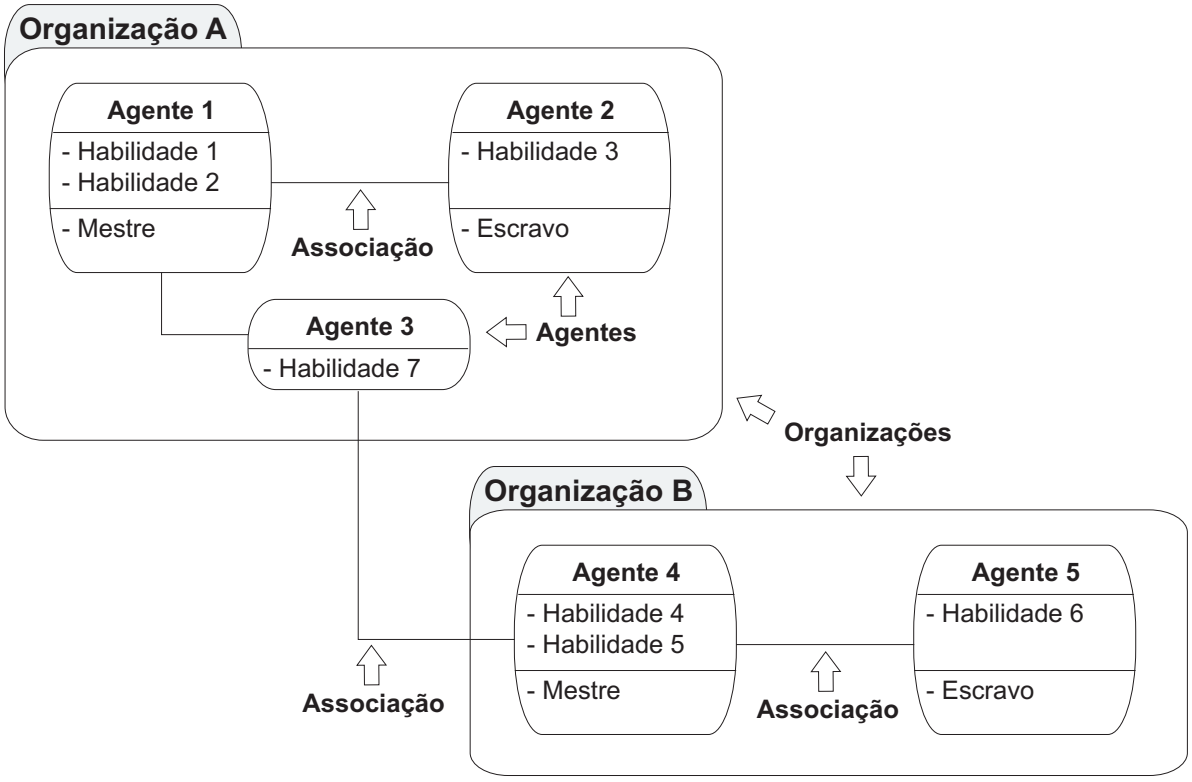

Figura A.5: Elementos do Diagrama de Agentes

Organização

Uma Organização é representada pelo símbolo ilustrado na Figura A.6. O mapeamento de contextos em organizações é do tipo 1 para 1. Sendo assim, cada organização representa um arranjo de agentes que compartilham determinado contexto do conhecimento.

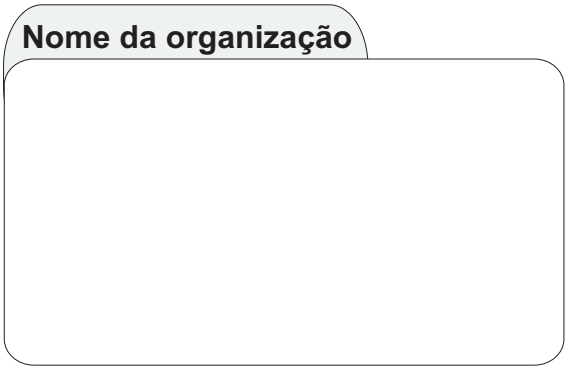

Figura A.6: Representação gráfica de uma Organização

Agente

Um Agente é representado no diagrama de agentes pelo símbolo ilustrado na Figura A.6. A representação gráfica do Agente é divida em três espaços onde são descritas as propriedades listadas a seguir.

- **Identificação do agente:** o nome do agente. Dentro de uma determinada organização, não pode haver dois agentes com o mesmo nome.
- **Habilidades:** uma lista das habilidades do agente. Um agente pode ter várias habilidades e uma habilidade pode ser contemplada por mais de um agente.
- **Papéis de interação:** uma lista dos papéis de interação os quais este agente representa. Um agente pode ter vários papéis de interação e um papel de interação pode ser representado por mais de um agente. A definição dos papéis de interação depende dos protocolos de interação que regem determinado agente [Jun03].

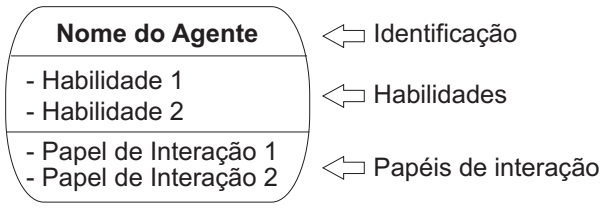

Figura A.7: Representação gráfica de um Agente

Associação

Uma associação é representada graficamente por um arco de ligação entre dois agentes. O sentido do arco define uma dependência das habilidades ou papéis de interação de um agente de origem em relação às habilidades ou papéis de interação do agente destino do arco, a ausência de sentido indica dependência mútua. Uma associação entre dois agentes representa a interação entre eles. Caso haja papéis de interação especificados nos agentes, esta associação representa a interação de acordo com o protocolo do qual fazem parte os papéis de interação. Na Figura A.8 são ilustradas duas associações entre agentes. A *Associação 1* representa a

interação entre os agentes 1 e 2 segundo as regras do protocolo Mestre-Escravo. Já a *Associação 2* representa a interação entre 1 e 3, sem protocolos pré-definidos [2]. As associações podem existir entre agentes de organizações diferentes.

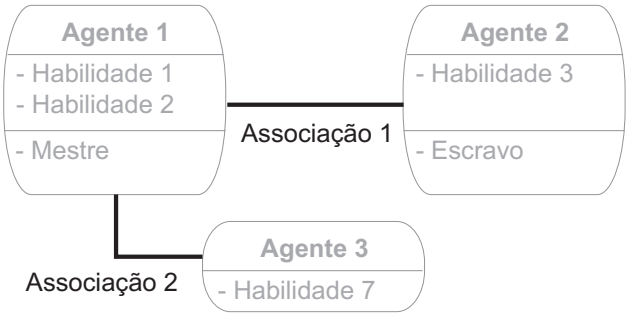

Figura A.8: Representação gráfica de uma Associação

Diagrama de Componentes

O diagrama de componentes tem foco na descrição dos componentes que pertencem a cada um dos agentes do sistema. Na Figura A.9 é apresentado o diagrama de componentes de COMPOR-L. A representação gráfica de um agente é diferente daquela do Diagrama de Agentes. Cada agente possui três espaços representativos dos três sistemas da arquitetura de agentes COMPOR: Sistema Inteligente(SI), Sistema Social(SS) e Sistema de Distribuição(SD). Os componentes são listados nestes espaços de acordo com o contexto de suas funcionalidades. Tem-se um diagrama de componentes para cada agente do sistema e assim um agente tem vários componentes, um componente pode estar presente em mais de um agente.

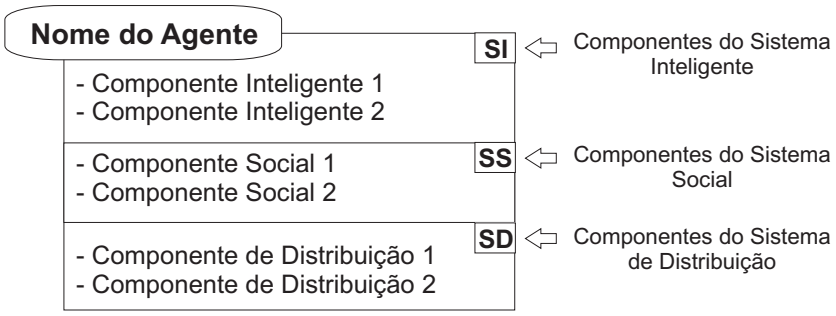

Figura A.9: Elementos do Diagrama de Componentes

[2] A nível de implementação pode significar uma invocação de método remoto, por exemplo.

Apêndice B

Análise comparativa de metodologias correlatas

Algumas metodologias para o desenvolvimento de sistemas multiagentes foram propostas e são analisadas com base nos critérios enumerados a seguir.

1. **Fases de desenvolvimento contempladas pela metodologia:** quais são as fases de desenvolvimento de software que a metodologia dá suporte? Tem-se uma seqüência de diretrizes que objetivam a produção de software?

2. **Dinamicidade no Sistema Multiagentes:** a metodologia considera a dinamicidade do sistema multiagentes resultante do refinamento dos requisitos como apresentado no Capítulo 2? Suas diretrizes refletem a preocupação com esta dinâmica?

3. **Escopo de aplicação:** quais são os tipos de aplicação às quais a metodologia se adequa ou é considerada mais indicada?

4. **Relação com as diretrizes COMPOR-M:** Relação entre a metodologia e as diretrizes COMPOR-M.

[Bur96]

Fases de desenvolvimento

A metodologia proposta por Burmeister [Bur96] contempla as fases de análise e projeto de software. Na fase de análise são definidos três modelos descritos a seguir.

1. **Modelo de agentes:** contém agentes e suas estruturas internas descritas em termos de estados mentais tais como objetivos, planos e crenças.

2. **Modelo de organizações:** especifica o relacionamento entre os agentes e tipos de agentes. Estes relacionamentos podem ser de herança entre tipos de agentes ou relações entre papéis dentro de uma organização.

3. **Modelo de cooperação:** descreve as interações entre os agentes. São definidos protocolos de interação, tipos de mensagens e padrões de cooperação entre os agentes.

Poucas considerações são feitas em relação à fase de projeto da metodologia, não ficando explícito se os modelos descritos já contemplam aspectos relacionados à esta fase. Após a construção dos modelos descritos anteriormente, não são feitas considerações sobre como transformar a descrição daqueles modelos em código. Diz-se apenas que a partir dos modelos gerados pode-se utilizar uma ferramenta que implemente o que os modelos descrevem.

Dinamicidade no Sistema Multiagentes

Nenhuma consideração é feita em relação à dinamicidade das organizações de agentes descritas no modelo de organizações. Nada é exposto em relação ao nível de adaptabilidade de um sistema desenvolvido com esta metodologia a possíveis refinamentos dos requisitos do sistema.

Escopo de aplicação

Nenhuma consideração é feita em relação ao escopo de aplicação da metodologia. O estudo de caso utilizado na referência pesquisada é um sistema de gerenciamento de um estacionamento de carros.

Relação com as diretrizes COMPOR-M

As diretrizes propostas por Burmeister servem como base para muitos dos outros trabalhos desta análise. Os modelos definidos porém são muito abstratos em relação à implementação do software. Considerando a definição de sistemas multiagentes, os modelos apenas determinam uma ordem em que são especificadas as propriedades desta definição - os agentes e a interação entre eles. Depois da especificação destas propriedades, não há um mapeamento para linguagens ou ferramentas que viabilizem a implementação. COMPOR-M utiliza a abordagem multiagentes mas tem como foco a implementação do software. Sendo assim, características de implementação são levadas em conta durante as fase de projeto, mapeando as habilidades em serviços providos por componentes de software.

Cassiopeia [DZ98]

Fases de desenvolvimento

Em Cassiopeia não são definidas fases de desenvolvimento mas um conjunto de diretrizes gerais são providas e apresentadas a seguir.

1. **Definição dos papéis individuais** - Identificação dos comportamentos que devem ser executados, agrupando-os em papéis. Agentes são definidos e os papéis são alocados aos agentes. A partir desta alocação de papéis a agentes, são identificados os tipos de agentes do sistema.

2. **Definição das dependências e relacionamento entre os papéis** - São definidas as dependências entre os papéis e os conseqüentes relacionamentos entre os agentes que contemplam estes papéis.

3. **Identificação dos grupos e dos papéis organizacionais** - Identificação dos potenciais grupos que possam se formar de acordo com os relacionamentos entre os agentes. Identificação dos papéis de interação dentro da organização de agentes formada. Em resumo, os protocolos de interação e os papéis dos agentes nestes protocolos são definidos.

Dinamicidade no Sistema Multiagentes

Nenhuma consideração é feita em relação à iteração no desenvolvimento, a qual poderia causar uma dinâmica no relacionamento entre os agentes.

Escopo de aplicação

Nenhuma consideração é feita em relação ao escopo da aplicação. O estudo de caso da referência pesquisada se refere ao domínio de futebol de robôs.

Relação com as diretrizes COMPOR-M

A relação com as diretrizes COMPOR-M se restringe às características inerentes a sistemas multiagentes - agentes e a interação entre eles. Se o sistema possui dois níveis de abstração, ou seja, nível de interação e nível de arquitetura interna de agentes, nada mais normal que existam diretrizes para a especificação destes dois níveis. Características de implementação não são levadas em conta nas diretrizes.

CoMoMAS [Gla97]

Fases de desenvolvimento

A metodologia CoMoMAS (*Conceptual Modelling of MultiAgent Systems*) tem foco na fase de análise, mais especificamente,na construção de modelos para sistemas multiagentes. Ela se baseia em CommonKADS [SWH+94] para prover um conjunto de diretrizes que auxiliam na construção destes modelos. Os seguintes modelos são definidos em CoMoMAS.

- **Análise funcional** - Identificação das tarefas que deveriam ser resolvidas pelo sistema multiagentes, definindo uma hierarquia de tarefas. Dependências entre as tarefas também devem ser identificadas. O resultado da análise é descrito em um Modelo de Tarefas.

- **Análise de requisitos** - Identificação dos requisitos do sistema e suas interdependências. Aspectos não funcionais devem ser levados em conta. Os resultados obtidos são descritos em um Modelo de Projeto.

- **Análise de competência** - Identificação as competências cognitivas e reativas que o sistema deve ter para resolver as tarefas descritas anteriormente. Estas competências são assumidas como sendo métodos para a resolução de problemas. O resultado desta análise é descrito em um Modelo Especialista.

- **Análise cooperativa** - Identificação dos protocolos e métodos de cooperação. Os resultados são descritos em um Modelo de Cooperação.

- **Análise social** - Identificação da organização e da arquitetura do sistema multiagentes. Os resultados são descritos em um Modelo de Sistema.

Dinamicidade no Sistema Multiagentes

Nenhuma consideração é feita em relação à iteração no desenvolvimento e à dinamicidade do sistema multiagentes.

Escopo de aplicação

A construção de modelos proposta é bem genérica. Não há restrição de domínio de aplicação da metodologia, ao menos na referência pesquisada.

Relação com as diretrizes COMPOR-M

O foco da metodologia é a construção de modelos, a qual é apontada como uma das fases da análise do problema. Em termos de implementação, utiliza-se uma linguagem semi-formal para a geração de código a partir dos modelos, mas não há considerações sobre a utilização em larga escala da abordagem. Também não fica claro como qual o nível de complexidade do problema com o qual esta geração de código é viável. A relação com as diretrizes COMPOR-M está na especificação de tarefas e protocolos de interação.

Gaia [WJK00; ZJW03]

Fases de desenvolvimento

Gaia dá suporte às fases de análise e projeto de aplicações multiagentes. Em relação à análise, não lida com atividades iniciais como a identificação e a especificação de requisitos. Aspectos de implementação não são considerados. A seguir são descritas as fases de análise e projeto da metodologia.

Fase de Análise

Na fase de análise, utiliza-se os requisitos ou objetivos do sistema para dividi-lo em sub-organizações. Para isto, é fornecido um conjunto de diretrizes que auxiliam no processo de identificação de uma sub-organização. Quatro modelos são especificados nesta fase e descritos a seguir.

- **Modelo do ambiente:** especifica-se o comportamento e os recursos do ambiente no qual o sistema funcionará.
- **Modelo de papéis:** os papéis relacionados ao sistema são identificados e descritos.
- **Modelo de interação:** especifica-se a interação entre os papéis do modelo de papéis.
- **Modelo de regras:** especifica-se um modelo de regras que devem reger a organização. Tanto as regras sobre a interação entre os papéis quanto axiomas que devem ter sempre sua condição satisfeita na organização são especificados.

Fase de Projeto

Todos os modelos gerados na fase de análise servem como entrada para a fase de projeto, a qual pode ser dividida em duas fases: projeto arquitetural e projeto detalhado.

- **Projeto arquitetural:** nesta fase os modelos de papéis e de interação são finalizados. Além disso, também é definida a topologia da estrutura organizacional do sistema.
- **Projeto detalhado:** com base no modelo de papéis os agentes são definidos. Em geral, o mapeamento é de um papel para um agente, mas isto não é obrigatório. No

modelo de papéis e na identificação dos agentes não foram identificados os serviços providos pelos agentes. Por isso, um modelo de serviços é proposto. Neste modelo são especificados blocos de atividades das quais os agentes serão executores.

Dinamicidade no Sistema Multiagentes

Nenhuma consideração é feita em relação à iteração no desenvolvimento com refinamento de requisitos. Gaia utiliza a abstração de organizações, cujo conceito está inerentemente relacionado à dinamicidade. Porém, esta dinamicidade não é levada em conta nas diretrizes.

Escopo de aplicação

Aplicável a qualquer sistema, inclusive sistemas abertos. Para isso utiliza a abstração de organizações.

Relação com as diretrizes COMPOR-M

Apesar de Gaia ser uma das mais completas metodologias para o desenvolvimento de sistemas multiagentes, existe uma grande distância entre os modelos definidos em suas diretrizes e a implementação do software. As diretrizes não guiam o desenvolvedor para produzir um software. Atributos essenciais de soluções de software como flexibilidade e reusabilidade não são considerados em nenhuma das diretrizes da metodologia. Além disso, cenários iterativos e evolutivos de desenvolvimento não são considerados, tais como a adição de funcionalidades e modificação das funcionalidades existentes.

As diretrizes COMPOR-M têm como principal característica guiar o desenvolvedor, utilizando a abstração de organizações de agentes, para a produção de software através do arcabouço COMPOR-F. Desde o primeiro momento da análise o desenvolvedor está sendo guiado à especificação de serviços providos pelos componentes que encapsulam código de software. Esta é uma das principais contribuições das diretrizes COMPOR-M em relação às metodologias existentes.

[KMJ95]

Fases de desenvolvimento

Como descrito no Capítulo 3, a metodologia proposta por Kendall *et al.* é uma das que se baseiam em conceitos da orientação a objetos. Nesta metodologia, as fases não são bem definidas, mas podem ser identificadas características das fases de análise - uma extensão dos casos de uso para agentes - e de projeto - definição da coordenação entre os agentes.

Dinamicidade no Sistema Multiagentes

Não são consideradas características dinâmicas do sistema e de iteração no desenvolvimento.

Escopo de aplicação

A metodologia tem como objetivo disponibilizar diretrizes para o desenvolvimento de sistemas de integração de empresas baseados em agentes.

Relação com as diretrizes COMPOR-M

A metodologia apresenta poucas diretrizes de desenvolvimento aplicada à concepção de software. Apenas uma extensão do diagrama de casos de uso para agentes é feita. Não há modelos de especificação dos agentes e, conseqüentemente, o mapeamento deste modelo em considerações de implementação. Não há ferramentas providas pela metodologia para auxiliar no desenvolvimento de software em larga escala. Ainda assim, foi uma das metodologias analisadas antes de conceber as diretrizes COMPOR-M, por isso, é descrita neste documento.

[KGR96]

Fases de desenvolvimento

Na metodologia proposta por Kinny *et al.* [KGR96] é disponibilizado um conjunto de modelos orientados a objetos especializados para o desenvolvimento de sistemas de agentes

BDI [RG95]. Os modelos são definidos sob dois pontos de vista: interno e externo. Na visão externa, o sistema é decomposto em agentes, com suas responsabilidades, serviços, informação requerida e relacionamentos externos. Estas informações são descritas em um Modelo de Agente e em um Modelo de Interação. Na visão interna, os elementos da arquitetura BDI são descritos para cada agente. Para isso, define-se um modelo de crenças, um modelo de objetivos e um modelo de planos para cada um dos agentes.

Dinamicidade no Sistema Multiagentes

Nenhuma consideração é feita em relação à dinamicidade do sistema multiagentes. Características de iteração do desenvolvimento que resultariam nesta dinamicidade não são abordadas.

Escopo de aplicação

A metodologia é voltada para a construção de sistemas de agentes BDI. Sendo assim é indicada para aplicações que possuam como requisito a utilização da arquitetura BDI.

Relação com as diretrizes COMPOR-M

Assim como a metodologia proposta por Burmeister [Bur96], esta metodologia serve como base para muitos dos outros trabalhos desta análise e seus modelos são muito abstratos em relação à implementação do software. Como descrito anteriormente em relação à metodologia de Burmeister, apenas a definição de sistemas multiagentes é contemplada nos modelos - os agentes e a interação entre eles. Após isto, não há considerações sobre a implementação ou utilização de ferramentas. Nas diretrizes COMPOR-M também são definidos os modelos externo e interno do agente porém, assim como nas outras metodologias mais recentes, estes modelos são especificado de forma menos abstrata durante a aplicação das diretrizes.

MADS [Sod01]

Fases de desenvolvimento

MADS contempla as fases de análise e projeto de sistemas, as quais são decompostas nas atividades descritas a seguir.

Análise

A fase de análise consiste das atividades de identificar objetivos, definir papéis e especificar interações entre estes papéis. Para cada uma destas atividades são gerados modelos, alguns deles baseados na metodologia MaSE [DWS01].

Projeto

Na fase de projeto, os papéis são estruturados em abstrações de agentes e um Diagrama de Agentes é gerado. Cada atividade de um agente pode ser detalhado em um Diagrama de Atividades, baseado no diagrama de atividades da metodologia Gaia [WJK00]. A arquitetura global do sistema é descrita em um Diagrama de Arquitetura. Este diagrama de arquitetura denota basicamente o relacionamento entre os papéis.

Dinamicidade no Sistema Multiagentes

Nenhuma consideração é feita sobre a dinamicidade do sistema multiagentes.

Escopo de aplicação

Não há escopo de aplicação definido. A princípio, MADS é aplicável à construção de qualquer tipo de sistema.

Relação com as diretrizes COMPOR-M

MADS não dá suporte ao projeto detalhado da aplicação multiagentes e não faz referências em suas diretrizes à definição de protocolos de interação entre os agentes. A arquitetura interna dos agentes também é pouco considerada e pouca atenção é dada às características

de implementação. Em relação às diretrizes COMPOR-M, MADS serviu como base para o estudo das metodologias Gaia [WJK00] e MaSE [DWS01], as quais estão presente nesta análise. Além disso, também serviu como ponto de partida para o estudo das características de análise e projeto de sistemas multiagentes. Tanto a infra-estrutura COMPOR quanto a metodologia MADS fazem parte do Projeto MaAE [Gir01].

MAS-CommonKADS [IGG98]

Fases de desenvolvimento

***M**ulti **A**gent **S**ystem-CommonKADS* é uma metodologia para a construção de modelos para Sistemas Multiagentes. Assim como a metodologia CoMoMAS analisada anteriormente, MAS-CommonKADS se baseia em CommonKADS para definir um conjunto de modelos para o desenvolvimento de SMA's. MAS-CommonKADS e CoMoMAS são semelhantes na definição das diretrizes e modelos de análise. Não há argumentos convincentes que comprovem o avanço de MAS-CommonKADS em relação à CoMoMAS [IGG98] neste contexto. Ainda assim, por representar um trabalho relacionado às diretrizes COMPOR-M, as diretrizes mais importantes referentes às fases de análise e projeto propostos em MAS-CommonKADS são descritos a seguir.

Análise

1. Modelagem de agentes - Identificam-se os papéis ou atores do sistema com base em uma análise das funcionalidades do sistema, através de casos de uso, por exemplo. Os papéis são atribuídos aos agentes que farão parte do sistema.

2. Modelagem de tarefas - Tarefas são decompostas hierarquicamente e são descritas em um modelo juntamente com seus nomes, descrições, estruturas etc. O modelo gerado é considerado importante em termos de documentação.

3. Modelagem de coordenação - Definição dos canais de comunicação e dos protocolos de coordenação.

4. Modelagem de conhecimento - Capacidades cognitivas dos agentes são definidas. O objetivo é semelhante à Modelagem do Especialista da metodologia CoMoMAS.

5. Modelagem de organização - Os papéis das organizações e as interações entre os agentes pertencentes à organização são definidos.

Projeto

1. Projeto de rede - A arquitetura de comunicação, compartilhamento de conhecimento entre os agentes é definida. Políticas sobre a organização também são definidas.

2. Projeto de agentes - Determina-se o projeto interno do agente de acordo com suas tarefas e objetivos.

3. Projeto de plataforma - Definição do software e do hardware necessário para executar o sistema.

Dinamicidade no Sistema Multiagentes

Nenhuma consideração é feita sobre a iteração no desenvolvimento com conseqüência na dinâmica no sistema multiagentes. A metodologia se restringe à definição dos modelos e não representa neles a característica de dinamicidade.

Escopo de aplicação

Nenhuma consideração é feita sobre restrições de aplicação da metodologia. O exemplo utilizado na referência pesquisada se refere ao domínio de planejamento de viagens.

Relação com as diretrizes COMPOR-M

Assim como citado anteriormente em relação à CoMoMAS, o foco da metodologia é a construção de modelos. Aspectos de implementação não são considerados. A relação com as diretrizes COMPOR-M está na especificação de tarefas e protocolos de interação.

MaSE [DWS01; DeL02]

Fases de desenvolvimento

***M**ultiagent **S**ystem **E**ngineering* contempla as fases de análise e projeto de sistemas multiagentes. MaSE possui o apoio da ferramenta agentTool [Lab03] para a automatização das fases de desenvolvimento, cujas diretrizes são descritas a seguir.

Análise

1. Identificar os objetivos a partir dos requisitos do sistema e descrevê-los em um Diagrama de Hierarquia de Objetivos.

2. Identificar os casos de uso e criar diagramas de seqüências para auxiliar na identificação do conjunto inicial de papéis e caminhos de comunicação.

3. Transformar os objetivos em um conjunto de papéis. Criar um Modelo de Papéis para descrever os papéis e as tarefas associadas a eles. Definir um Modelo de Tarefas Concorrentes para cada tarefa para identificar o comportamento do papel.

4. Definir um Modelo organizacional identificando as regras a serem impostas sobre cada um dos papéis. Este modelo foi adicionado à especificação de MaSE em versões mais recentes [DeL02].

Projeto

1. Definir um Diagrama de Classes de Agentes com base nos papéis de análise.

2. Definir um conjunto de tarefas organizacionais e atribuí-las às classes de agentes.

3. Definir novos papéis organizacionais e atribuí-los às classes de agentes.

Dinamicidade no Sistema Multiagentes

A dinamicidade no sistema multiagentes é considerada nas novas versões de MaSE, tal como apresentado em [DeL02].

Escopo de aplicação

Não há um escopo definido para a aplicação de MaSE. Em sua nova versão, MaSE é adequada inclusive para sistemas abertos.

Relação com as diretrizes COMPOR-M

Assim como Gaia, MaSE evoluiu suas diretrizes e acoplou o conceito de organização para, a partir daí, adequar-se ao desenvolvimento de sistemas multiagentes com característica dinâmica. Apesar deste avanço, não há considerações de implementação para que os artefatos gerados em análise e projeto se tornem código. Sendo assim, a relação entre as diretrizes COMPOR-M e a nova versão de Gaia, apresentada anteriormente nesta análise, também é válida em relação à MaSE.

MASSIVE [Lin00a; Lin00b; Lin00c]

Fases de desenvolvimento

***M**ultiAgent **S**ystem**S** **I**terative **V**iew **E**ngineering* é um processo que define um conjunto de passos necessários ao desenvolvimento de sistemas multiagentes. Não há uma seqüência de fases de desenvolvimento, em vez disto, MASSIVE baseia-se no conceito de visões para conceber o sistema, as quais são descritas a seguir.

- **Tarefas** - Os aspectos funcionais do sistema são analisados e organizados em uma hierarquia de tarefas. Além disso também são identificados os requisitos não-funcionais.
- **Ambiente** - O ambiente no qual o sistema vai funcionar deve ser analisado.
- **Papéis** - São determinados os papéis dos agentes e uma arquitetura que seja capaz de contemplar estes papéis.
- **Interação** - Descrevem-se as interações entre os agentes.
- **Sociedade** - Descreve-se como os agentes devem ser agrupados.

- **Arquitetural** - A arquitetura interna do agente e do sistema são descritos. A arquitetura interna do agente é descrita apenas em termos de propriedades, não em relação a aspectos de implementação. Na definição da arquitetura do sistema, por sua vez, são citados apenas alguns modelos arquiteturais clássicos que poderiam ser utilizados [SG96].

- **Sistema** - São identificados os aspectos que afetam todas as outras visões, como por exemplo, gerenciar a interface do usuário que controla a interação entre o sistema e o usuário.

Dinamicidade no Sistema Multiagentes

MASSIVE leva em conta a iteração do desenvolvimento para dar suporte à característica dinâmica do sistema. Porém, isto é apenas citado, ou seja, não são apresentadas que garantam esta dinâmica em relação à implementação do software.

Escopo de aplicação

MASSIVE não define um escopo de aplicação, ou seja, pode ser aplicado ao desenvolvimento de sistemas multiagentes em qualquer domínio.

Relação com as diretrizes COMPOR-M

MASSIVE usa um processo iterativo que considera o refinamento dos requisitos do software. Entretanto, em todas as visões e etapas propostas nesta metodologia, apenas as características a serem identificas no sistema são descritas. MASSIVE tem um conjunto de diretrizes abrangente mas muito genérico. Não há considerações sobre como mapear agentes em código ou como acoplar as visões à ferramentas de desenvolvimento. Como citado anteriormente, as diretrizes COMPOR-M foram concebidas objetivando o software e portanto contempla este mapeamento.

MESSAGE/UML [CCG+01]

Fases de desenvolvimento

MESSAGE/UML é uma metodologia que contempla as fases de análise e projeto, segundo descrito em [CCG+01]. Porém, nesta mesma referência apenas considerações de análise são feitas e outras referências não foram encontradas durante o levantamento bibliográfico. Sendo assim, descreve-se a seguir, as visões que

Dinamicidade no Sistema Multiagentes

A abstração de organizações é utilizada, porém não a nível de projeto e implementação.

- **Visão de Organização -** São definidos os agentes, papéis, recursos e interações entre eles.
- **Visão de Objetivo/Tarefa -** São definidos os objetivos, tarefas e dependências entre eles.
- **Visão de Agente/Papel -** São especificados cada um dos papéis e tarefas atribuídos a cada agente.
- **Visão de Interação -** São especificados os detalhes de cada interação - informações que são trocadas, eventos que podem ser disparados etc.
- **Visão de Domínio -** Definição dos conceitos do domínio do problema, os quais são relevantes à especificação do sistema.

Para cada uma das visões são gerados diagramas, alguns deles com base em AUML [OPB00].

Escopo de aplicação

Não há um escopo de aplicação definido para a utilização de MESSAGE/UML.

Relação com as diretrizes COMPOR-M

Uma vez que não foram encontradas referências sobre a fase de projeto da metodologia, a relação com as diretrizes COMPOR-M se restringe à existência de uma visão de domínio de conhecimento, assim como MAS-CommonKADS e CoMoMAS. O restante das visões possuem versões similares em outras metodologias e já foram comentadas anteriormente.

Prometheus [PW02]

Fases de desenvolvimento

Na metodologia Prometheus são definidas três fases onde são produzidos artefatos que são utilizados para a geração do esqueleto do código, teste e depuração. Tais fases são descritas a seguir.

- **Fase de Especificação do Sistema** - São identificadas as funcionalidades básicas, suas entradas, saídas e quaisquer fontes de dados compartilhadas.
- **Fase de Projeto Arquitetural** - São utilizadas as entradas e saídas da fase anterior para determinar os agentes do sistema e suas interações.
- **Fase de Projeto Detalhado** - A estrutura interna do agente necessária à resolução de suas tarefas é especificada.

Dinamicidade no Sistema Multiagentes

Nenhuma consideração é feita sobre a dinamicidade do sistema multiagentes. Prometheus não considera a abstração de organizações de agentes.

Escopo de aplicação

Não há um escopo definido para a aplicação para Prometheus.

Relação com as diretrizes COMPOR-M

Assim como nas diretrizes COMPOR-M, a montagem dos agentes a nível de implementação é feita através de uma ferramenta. Esta ferramenta gera o esqueleto de código a partir de uma especificação de modelos definidos por Prometheus. A definição dos agentes em Prometheus é baseada no agrupamento de funcionalidades, não levando em conta o particionamento do domínio, ou seja, o conhecimento relacionado ao agente. Este diferencial é contrário à própria definição de agentes, ao menos de acordo com todas as outras referências aqui analisadas.

ROADMAP [RPS02]

Fases de desenvolvimento

***R**ole **O**riented **A**nalysis and **D**esign for **M**ulti-**A**gent **P**rogramming* é uma extensão da metodologia Gaia para a análise e projeto de sistemas multiagentes. A principal motivação apresentada por esta metodologia é que Gaia não aborda a dinamicidade do sistema multiagentes. Esta metodologia considerava a versão antiga da metodologia Gaia [WJK00] onde o conceito de organizações ainda não havia sido acoplado. Alguns modelos são acrescentados aos da metodologia Gaia e são apresentados a seguir.

Análise

- **Modelo de casos de uso** - São descritos os requisitos dos sistemas. Conceitos de ROADMAP como papéis e zonas podem ser utilizados. É uma extensão do diagrama de casos de uso da orientação a objetos.
- **Modelo do ambiente:** - Descrição do ambiente e dos objetos que pertencem a ele. O conceito de “Zona” é utilizado para particionar o ambiente. Em cada zona podem ser identificados atributos como objetos estáticos, objetos, restrições e fontes de incerteza.
- **Modelo de conhecimento:** - Descrição do domínio de conhecimento no qual se insere o sistema.

- **Modelo de papéis revisado:** - O modelo de papéis definido em Gaia é redefinido e consiste de dois artefatos: uma hierarquia de papéis; e um conjunto de descrições para cada um dos papéis da hierarquia.

Projeto

- **Modelos de papéis e modelos de agentes** - O modelo de papéis é refinado e é definido um modelo de agente similar ao definido em Gaia.

Dinamicidade no Sistema Multiagentes

A dinamicidade do sistema é tratada mas não a nível de implementação.

Escopo de aplicação

Assim como na metodologia Gaia, não é definido um escopo de aplicação para a metodologia.

Relação com as diretrizes COMPOR-M

Apesar de considerar a dinamicidade do sistema multiagentes, ROADMAP não guia o desenvolvedor à implementação do software. A distância entre o resultado do projeto e a implementação é grande, assim como na metodologia Gaia. ROADMAP se propõe a acrescentar o conceito de dinamicidade à metodologia Gaia, o que já havia sido feito na versão mais recente desta metodologia [ZJW03]. A relação entre as diretrizes COMPOR-M e a nova versão de Gaia, apresentada anteriormente nesta análise, é válida em relação à ROADMAP.

SODA [Omi00]

Fases de desenvolvimento

***S**ocieties in **O**pen and **D**istributed **A**gent spaces* é uma metodologia para a análise e projeto de aplicações multiagentes que possui como principais entidades de abstração a sociedade

de agentes e o ambiente no qual a sociedade está inserida. Os modelos e diretrizes definidos em SODA são descritos a seguir.

Análise

- **Modelo de papéis** - Os objetivos da aplicação são modelados em termos das tarefas a serem cumpridas, as quais estão associadas a papéis e grupos.
- **Modelo de recursos** - O ambiente da aplicação pe modelado em termos dos serviços disponíveis, os quais estão associados a recursos.
- **Modelo de interação** - A interação entre os papéis, grupos e recursos é modelada em termos dos protocolos de interação, expressos como informação requerida e provida pelos papéis e recursos. Além disso, são descritas regras de interação que governam os grupos.

Projeto

- **Modelo de agentes** - Papéis individuais e sociais são mapeados para classes de agentes.
- **Modelo de sociedade** - Grupos são mapeados em sociedades de agentes, os quais são projetados e organizados através de abstrações de coordenação.
- **Modelo de ambiente** - Recursos são mapeados em classes de infra-estrutura e associados à topologias.

Dinamicidade no Sistema Multiagentes

De acordo com a abstração de sociedades, SODA dá suporte à dinamicidade nos relacionamentos entre os agentes. Da mesma forma, permite a entrada de novos agentes no sistema. Porém, isto não é tratado a nível de implementação.

Escopo de aplicação

Não há escopo de aplicação definido para a metodologia SODA.

Relação com as diretrizes COMPOR-M

A abstração de sociedades utilizada por SODA é similar à abstração de organizações proposta na metodologia Gaia [ZJW03]. A única diferença é que a noção de papéis em Gaia é considerada uma derivação de tarefas e serviços em SODA. Esta pode ser uma boa abordagem para adequar a noção de habilidades das diretrizes COMPOR-M à noção de papéis. Assim como nas outras metodologias, características de implementação não são levadas em conta.

TROPOS [MKC01; CKM01; GMP02]

Fases de desenvolvimento

TROPOS é uma metodologia para análise de requisitos de um sistema multiagentes, com base na noção de atores, objetivos e dependências. Para isso, são definidos diversos diagramas para a especificação dos atores de um sistema, seus objetivos e as dependências entre eles. Após esta definição, espera-se obter um projeto de agentes mais fiel à especificação do problema.

Dinamicidade no Sistema Multiagentes

A dinamicidade do sistema é citada mas não a nível de projeto e implementação.

Escopo de aplicação

Não há um escopo definido para a aplicação de TROPOS.

Relação com as diretrizes COMPOR-M

Uma vez que TROPOS tem como meta apenas a análise dos requisitos do sistema, não há uma relação direta com as diretrizes COMPOR-M. Como perspectiva futura poderia ser feito um acoplamento da análise de *early-requirements* de TROPOS às diretrizes COMPOR-M. Desta forma, analistas de requisitos TROPOS poderiam mapear a análise em um sistema de software com a infra-estrutura COMPOR.

Na Tabela B.1 é apresentado o resultado da análise comparativa das metodologias. Os parâmetros são os mesmos descritos anteriormente, sendo as fases de desenvolvimento representadas por Análise(A), Projeto(P), Implementação(I) e Testes(T). Caso a metodologia não apresente a divisão em fases mas possua outro tipo de divisão, este atributo será representado por (O). Em relação ao escopo de aplicação, considera-se uma metodologia como "escopo geral" caso ela se adeque a qualquer aplicação multiagentes. A coluna referente à dinamicidade será marcada caso a metodologia a considere em análise ou projeto. Nenhuma das metodologias analisadas leva em conta a dinamicidade a nível de implementação.

Metodologias	**Características**		
—	Fases	Dinamicidade	Escopo geral
[Bur96]	(A)(P)		•
Cassiopeia	(O)		•
CoMoMAS	(A)		•
Gaia	(A)(P)	•	•
[KMJ95]	(A)(P)		
[KGR96]	(A)(P)		
MADS	(A)(P)		•
MAS-CommonKADS	(A)(P)		•
MaSE	(A)(P)	•	•
MASSIVE	(O)	•	•
MESSAGE-UML	(A)(D)		•
Prometheus	(O)		•
ROADMAP	(A)(P)	•	•
SODA	(A)(P)	•	•
TROPOS	(A)	•	•

Tabela B.1: Tabela comparativa das metodologias para a construção de sistemas multiagentes

Apêndice C

Análise comparativa de arcabouços correlatos

Neste apêndice será apresentada uma análise dos arcabouços para a construção de sistemas multiagentes(SMA's). A estrutura da análise é composta pelos tópicos descritos a seguir.

1. **Resumo:** uma breve descrição do arcabouço.
2. **Arquitetura:** apresentação do projeto arquitetural do arcabouço, seus principais componentes e restrições de projeto.
3. **Implementação:** tecnologias e métodos utilizados para a implementação: linguagens de programação e modelagem, processos de desenvolvimento, protocolos etc.
4. **Instanciações:** exemplos de aplicações construídas com base no arcabouço.
5. **Relação com COMPOR-F:** Relação entre o arcabouço e o arcabouço COMPOR-F.

Brainstorm/J [ZA00]

Resumo

A arquitetura que dá suporte a este arcabouço permite que objetos simples sejam estendidos como agentes. Esta extensão é suportada por meta-objetos [Mae87] que representam um modo flexível e adaptado para compor componentes. A abordagem é fundamentada na idéia

de que um SMA pode ser considerado um sistema orientado a objetos que possui um meta-sistema associado [AP97]. Este meta-sistema incorpora o comportamento de agente a um simple objeto. Assim, um agente é considerado um objeto com capacidades de comunicação, percepção, reação e deliberação.

Para prover capacidades de agente a objetos, utilizam-se meta-objetos [Mae87]. Meta-objetos são objetos especiais que têm a habilidade de interceptar as invocações de métodos, alterando assim a sua execução. Cada objeto, então, possui muitos meta-objetos associados, dependendo das características de agente requeridas para aquele objeto. Desta forma, arquiteturas híbridas são possíveis, uma vez que objetos diferentes podem ter associações diferentes com meta-objetos.

Arquitetura

Como descrito anteriormente, tem-se dois níveis principais na arquitetura: o nível de objetos e o nível de agentes (meta-objetos). As características de reatividade e deliberação foram concebidas através da percepção do ambiente. Uma vez que os meta-objetos podem interceptar as invocações de métodos dos objetos do nível base, podem perceber as alterações que ocorreram no ambiente e reagir ou planejar uma ação em resposta.

Implementação

Brainstorm/J foi implementado usando JavaLog [AZI99], que é uma linguagem multi-paradigma baseada em Java e Prolog. O suporte de meta-objetos para Java foi baseado em LuthierMOPS [CP99].

Instanciações

Dois SMA's foram desenvolvidos: *Forklifts* e n-rainhas. O primeiro consiste de um conjunto de "carros carregadores de mercadoria" com o objetivo de arranjar um conjunto de caixas. Este sistema foi usado para testar comportamentos como planejamento reativo e heurísticas para a resolução de conflito.

O segundo sistema implementado foi uma solução multiagentes para o problema clássico das n-rainhas. Neste sistema, cada agente representa uma rainha com o objetivo único de encontrar uma posição segura. Neste sistema, os agentes se comunicam via KQML [FFMM94].

Relação com COMPOR-F

A abordagem de meta-objetos permite que os papéis desempenhados pelos agentes tenham um caráter dinâmico. Sendo cada papel um meta-objeto, pode-se alterar dinamicamente o conjunto de papéis de um agente. Esta possibilidade é similar à possibilidade de alteração de serviços provida por COMPOR-F. Porém, a linguagem utilizada para a implementação com *Brainstorm/J* não é largamente utilizada e a utilização de reflexão computacional [1] torna mais complexa a construção de agentes, pois é necessário lidar com dois níveis de implementação: o nível de objetos e o nível de meta-objetos. Além disso, a abordagem de objetos para a implementação das funcionalidades dos agentes traz menos reusabilidade que a abordagem de componentes implementada por COMPOR-F.

DECAF [GD99]

Resumo

Distributed, Environment-Centered Agent Framework é uma ferramenta que permite a construção de SMA's. DECAF provê os serviços de suporte necessários à concepção de agentes inteligentes: comunicação, planejamento, escalonamento, monitoramento de execução, coordenação, aprendizado e auto-diagnóstico. Neste apêndice, o foco estará sobre o arcabouço que dá suporte à implementação dos agentes da ferramenta DECAF.

Arquitetura

A arquitetura é composta por três componentes: um servidor de nomes de agentes (ANS); um programa de agente ou arquivo de plano; e o modelo do arcabouço. O ANS é um servidor de nomes comum, como um DNS. Os agentes que entram no sistema se registram junto ao ANS e a comunicação é feita através da recuperação deste registro ou endereço, como num catálogo. O arquivo de plano é a programação do agente. Um agente consiste de um conjunto de objetivos e uma coleção de ações que podem ser planejadas e executadas em busca dos seus objetivos. Cada um destes objetivos e a programação para alcançar os objetivos são feitas no arquivo de plano.

[1] A idéia de meta-objetos está inserida no contexto de reflexão computacional.

Todos os requisitos de interação entre os agentes são retirados da responsabilidade do programador. O desenvolvedor não precisa escrever código de programação, nem se preocupar com a invocação de métodos num mesmo agente, ou analisar e escalonar mensagens de entrada. Não há uma interface do programador da aplicação (API) para programar sobre a arquitetura DECAF. Todos os fluxos de dados dos agentes são descritos no arquivo de plano.

Implementação

A linguagem Java foi utilizada para a implementação do framework e da ferramenta DECAF. O programador codifica as ações dos agentes nos arquivos de plano como já foi descrito anteriormente.

Instanciações

Foram feitos testes em uma sala de aula com quinze estudantes que não conheciam a programação de agentes. Em quatro semanas, vinte agentes haviam sido desenvolvidos, os quais deram origem a duas pequenas aplicações.

Aplicações mais robustas também foram desenvolvidas: *Virtual Food Court*(VFC) era representado por sete agentes diferentes, o que foi útil para o teste de interações e técnicas de negociações entre os agentes; e *BigAgent*, em que foi desenvolvido um plano com mais de cem possíveis escalonamentos para completar uma tarefa.

Relação com COMPOR-F

Assim como outros arcabouços analisados, o arcabouço DECAF implementa a infra-estrutura de interação entre os agentes de forma fixa. Sendo assim, novos protocolos e linguagens de interação entre agentes não podem ser implementadas ou, pelo menos, não são dados subsídios que facilitem esta implementação. Além disso a abordagem baseada em componentes de COMPOR-F objetiva maior reusabilidade e flexibilidade que a abordagem orientada a objetos de DECAF.

DSAP [BFP00]

Resumo

Distributed Software Agent Plataform é uma plataforma de desenvolvimento de aplicação que usa a tecnologia de agentes em ambientes distribuídos, implementados em Java/RMI. A plataforma consiste de um grupo escalável de servidores usados por aplicações de muitos domínios: comércio eletrônico, busca e educação à distância.

Arquitetura

A arquitetura da plataforma é dividida em dois módulos: servidor do agente e arcabouço do agente.

O servidor do agente provê a infra-estrutura e os serviços necessários às aplicações multi-plataformas e distribuídas em SMA's. Ele manipula requisições da aplicação e de outros servidores de agentes e contém os seguintes componentes: comunicação, gerenciamento, registro e transporte.

O arcabouço do agente fornece as classes necessárias para a criação dos agentes que irão realizar suas atividades num ambiente distribuído. A classe Agente reutiliza o padrão de projeto Abstract Factory [GHJV95] que dá suporte à criação de famílias de objetos dependentes relacionados sem especificar suas classes concretas. Define um modelo de tipos de agentes e estratégias para cada um destes tipos. Segue o método Catalysis de desenvolvimento [DW03].

Implementação

Utiliza Java como linguagem de programação, Java/RMI para comunicação e o método de desenvolvimento Catalysis.

Instanciações

Foi desenvolvida uma aplicação para monitoramento de computadores (ReMA - Remoting Monitoring Application). Através desta ferramenta os agentes podem ser criados para monitorarem computadores remotamente.

Relação com COMPOR-F

O arcabouço DSAP é mais direcionado à mobilidade que a características de interação entre os agentes. Assim como o arcabouço COMPOR-F, DSAP possui seu projeto baseado em padrões de projeto [GHJV95]. A escalabilidade do arcabouço para a adição de novos protocolos e linguagens de interação não é citada. Tal escalabilidade é garantida por COMPOR-F, que provê flexibilidade para a definição de componentes que encapsulam todas as funcionalidades dos agentes, inclusive as de interação.

FraMAS [AA00]

Resumo

O projeto geral de FraMAS tem como proposta para modelar SMA's a abordagem de ambientes. Cada ambiente manipula agentes. Estes agentes interagem com outros agentes para alcançar seus objetivos. Agentes têm a capacidade de se mover de um ambiente para outro. Para fazer isto o arcabouço provê primitivas de agente e primitivas de ambiente. As primitivas de agente são *start* (execução padrão) e *stop* (estado seguro, onde o código e os dados estão salvos). As primitivas de ambiente são *requestToTransfer* (o ambiente local informa ao ambiente remoto sobre seu desejo de transferir um agente) e *beginTransfer* (o ambiente remoto marca o início da transferência).

Arquitetura

Para o projeto dos agentes, utiliza-se o padrão *Decorator* [GHJV95], que permite adicionar funcionalidade a objetos dinamicamente. Define-se então um comportamento básico (interação, armazenamento de dados etc) e "decoradores" para este comportamento com as características avançadas (planejamento, deliberação etc). Os comportamentos reativo e deliberativo são obtidos utilizando, respectivamente, *listeners* (Padrão *Observer* [GHJV95]) e Raciocínio Baseado em Casos (RBC) [Wat97].

Implementação

A linguagem de implementação é Java/Sun e a comunicação é feita utilizando RMI. A linguagem de comunicação entre agentes utilizada é KQML [FFMM94].

Instanciações

Duas instanciações do arcabouço foram feitas: um SMA formado por agentes de agendamento de encontros e o sistema *Forks*. No primeiro sistema, cada agente de agendamento pertence a um usuário e tem a funcionalidade de aprender as preferências deste usuário e negociar a agenda com outros agentes. As preferências do usuário são aprendidas com base em exemplos. Estes exemplos são obtidos diretamente da interação usuário-agenda e salvos em uma base de casos. Então, utiliza-se RBC para identificar similaridades entre os casos de agendamento definindo a preferência do usuário. O segundo SMA é composto por agentes que interagem para arranjar um conjunto de caixas (*forklifts*).

Relação com COMPOR-F

Apesar de estar baseado em padrões de projeto, o arcabouço é pouco flexível devido ao abuso da herança [GHJV95]. O comportamento deliberativo do agente é definido como um módulo de Raciocínio Baseado em Casos(RBC) [Wat97]. Este módulo faz parte do projeto de um agente considerado deliberativo. Porém, RBC pode não ser adequado em determinados domínios de aplicação de SMA's. Logo, a forma de raciocínio de um agente deveria ser genérica e representada como um *hot spot* [FJS00] do arcabouço. Uma vez que as funcionalidades dos agentes em COMPOR-F são implementadas pelos componentes, o módulo RBC do arcabouço FraMAS pode ser implementado como um componente de COMPOR-F. Desta forma, obtém-se a possibilidade de prover um agente com o componente RBC, por demanda. O mesmo acontece em relação a tecnologias e protocolos de interação, os quais são fixos em FraMAS, que possui uma pobre definição de ganchos [FJS00].

JACK [BRHL99; HRHL01]

Resumo

O arcabouço JACK tem como característica diferencial em relação aos outros arcabouços a co-existência de agentes e objetos no mesmo espaço de projeto e programação. A justificativa dada a esta abordagem é que a maioria dos sistemas já implantados ou legados, podem necessitar de características de agentes em suas implementações orientadas a objetos, então agentes podem trocar mensagens com objetos e tornar características deliberativas presentes

nestes sistemas. Em JACK, agentes são acessórios da linguagem orientada a objetos, por isso, a linguagem Java é redefinida para que os termos agentes e eventos (para percepção do ambiente) sejam incorporados à linguagem. O programador Java deverá então utilizar esta nova sintaxe e o compilador JACK compilará o código fazendo a adaptação para o compilador Java.

Arquitetura

A arquitetura do arcabouço é baseada no modelo BDI [RG95]. Agentes possuem crenças e desejos, ou objetivos em relação a outros agentes, e possuem intenções ou planos para que seus objetivos sejam alcançados. A arquitetura é orientada a eventos e o comportamento pró-ativo tem base na percepção do ambiente através da troca de mensagens entre agentes.

Implementação

A implementação do arcabouço é feita em Java. A instanciação do arcabouço, ou seja, a construção de sistemas com JACK é feita usando uma extensão de Java, com alguns novos termos para a implementação de agentes.

Instanciações

Duas aplicações foram desenvolvidas com JACK: um sistema de suporte à decisão e uma aplicação em simulação de defesa. No primeiro sistema, JACK é usado para a construção dos agentes de planejamento que, dada uma entrada com informações geográficas sobre um campo de batalha, exibe um conjunto de planos possíveis para um comandante escolher o que melhor se adequa ao momento, como um suporte à sua decisão. O segundo é uma aplicação de modelagem do comportamento humano baseada na análise do comportamento de agentes.

Relação com COMPOR-F

O principal problema de JACK em relação a outros arcabouços é que ele define uma linguagem de programação de agentes como uma extensão da linguagem Java. A evolução da arquitetura proposta está ligada diretamente ao compilador JACK, que faz com que o código

da extensão Java funcione em uma máquina virtual comum. Sendo assim, extensões no arcabouço devem ser refletidas no compilador JACK. Como visto anteriormente, o desenvolvedor que utiliza COMPOR-F implementa as funcionalidades dos agentes como componentes. Isto implica dizer que qualquer funcionalidade pode ser um componente e, portanto, o arcabouço pode evoluir em funcionalidade de acordo com os componentes. Esta flexibilidade foi um dos principais requisitos na construção de COMPOR-F.

JADE [BRP99]

Resumo

Java Agent DEvelopmemt Framework é um arcabouço para o desenvolvimento de aplicações multiagentes compatível com as especificações FIPA (*Foundation of Intelligent Phisical Agents*). FIPA é uma fundação que tem como objetivo produzir especificações de tecnologias de agentes genéricas, visando uma padronização na definição de sistemas de agentes [ON98]. Para isso, FIPA define um conjunto de regras para a interoperabilidade e gerenciamento de agentes em uma sociedade, deixando o comportamento externo de cada agente a cargo do programador da aplicação. Desta forma, busca-se a interoperabilidade entre diferentes plataformas de agentes, uma vez que estas respeitarão as mesmas regras de interação. É com base nesta motivação que foi concebido o arcabouço JADE, o qual possui as seguintes características:

- uma plataforma de agentes compatível com FIPA que inclui o sistema de gerenciamento de agentes (AMS), o facilitador de diretório (DF) e o canal de comunicação de agentes (CCA);
- plataforma de agentes distribuída;
- serviço de nomes e registro por domínio através do DF;
- mecanismo de transporte e interface para enviar/receber mensagens para/de outros agentes;
- transporte diferenciado de mensagens ACL dependendo da localização dos agentes, visando melhor desempenho, através do CCA;

- biblioteca de protocolos de interação FIPA;
- interface gráfica de usuário para gerenciar agentes e plataformas através de monitoramento e geração de *logs*, através do AMS.

Arquitetura

A arquitetura de JADE é composta pelos sistemas ACC, AMS e DF, descritos anteriormente. Toda a comunicação entre agentes é feita através de transporte de mensagens, cuja linguagem é FIPA - ACL [fIPA02]. Entre plataformas JADE, o transporte de mensagens é feito através de RMI, para a comunicação com outras plataformas, utiliza-se CORBA IIOP [Gro03]. Todos os agentes JADE utilizam o objeto *ACLMessage* como uma mensagem, sendo assim, uma invocação sobre CORBA é traduzida para um objeto *ACLMessage* para poder ser processada por JADE.

Além de prover a infra-estrutura para o gerenciamento da comunicação entre agentes, JADE disponibiliza também um modelo de execução de agentes baseado na composição de comportamentos. Cada agente possui um conjunto de comportamentos associado e alguns destes comportamentos foram implementados como padrões. Há também integração com a máquina de inferência de conhecimento JESS [FH02].

Implementação

A implementação da plataforma foi feita utilizando a linguagem Java. RMI e CORBA foram utilizados para a camada de comunicação.

Instanciações

JADE já é amplamente utilizado. Dentre os sistemas desenvolvidos com JADE podem ser citados um sistema de entretenimento audio-visual baseado em agentes, um assistente pessoal de viagens e um assistente para o mercado de ações.

Relação com COMPOR-F

JADE é compatível com FIPA e possui uma infra-estrutura de comunicação que provê transporte de mensagem, gerenciamento, serviço de nomes e registro de agentes e interopera-

bilidade entre diferentes plataformas de agentes. Além disso, possui integração com uma máquina de inferência de conhecimento - JESS [FH02]. A principal diferença entre JADE e COMPOR-F está na flexibilidade da infra-estrutura. COMPOR-F encapsula as funcionalidades em componentes, permitindo que o desenvolvedor defina as tecnologias e linguagens de interação a serem utilizadas. Esta infra-estrutura de componentes também permite que componentes de planejamento sejam definidos, permitindo que sejam reutilizados posteriormente. Em JADE, a definição de funcionalidades do agente que não dizem respeito à interação são muito genéricas. O desenvolvedor não possui modelos de reutilização, a não ser a nível de classe - a classe Agent, ou seja, JADE pode ser visto como um arcabouço para a interação entre objetos que interagem via FIPA-ACL. Não há facilidades no planejamento ou definição de comportamento dos agentes, isto fica a cargo do desenvolvedor.

JAF [VHL01]

Resumo

Java Agent Framework é um arcabouço baseado em componentes para a construção de SMA's. A construção de um agente é feita através da composição dos componentes disponibilizados por JAF ou, caso necessário, de novos componentes construídos pelo desenvolvedor. Também provê um ambiente de simulação de SMA's para verificação da execução e funcionamento da sociedade. Os componentes disponibilizados por JAF seguem a arquitetura *Java Beans* [Mic97] o que os torna compatíveis com ferramentas já existentes de composição de componentes.

Arquitetura

A arquitetura de JAF é baseada na composição de componentes e a comunicação através de eventos que causa dependência entre eles. Agentes são compostos de componentes e estes encapsulam toda a funcionalidade, inclusive de interação, destes agentes. Muitos componentes são disponibilizados para a construção de agentes, dentre eles, componentes de comunicação, escalonamento, serviço de diretório e resolvedor de tarefas.

Implementação

A implementação foi feita em Java e os componentes foram construídos seguindo o modelo *Java Beans* [Mic97].

Instanciações

JAF foi utilizado para o desenvolvimento de três projetos:um ambiente de rede de sensores distribuídos, onde agentes foram testados no simulador e depois migraram para o mundo físico real dos sensores; um projeto de casa inteligente, onde agentes foram colocados em um ambiente e coordenavam suas ações em busca de um objetivo, com base nesta coordenação foi feita uma análise comparativa dos protocolos de interação entre agentes; e um protótipo de um ambiente para o domínio Produtor, Consumidor e Transportador (PCT), onde existem três tipos de agentes (produtores, consumidores e transportadores) que funcionam e raciocinam de forma diferente em busca de objetivos diferentes.

Relação com COMPOR-F

Como apresentado no Capítulo 5, os agentes do arcabouço baseado em componentes JAF [VHL01] são formados por componentes. A principal diferença está no modelo de componentes *JavaBeans* no qual JAF se baseia. *JavaBeans* não provê independência entre os serviços disponibilizados e requeridos pelos componentes funcionais, como apresentado no Capítulo 4. Outro diferencial do arcabouço COMPOR-F é a interação assíncrona. Além disso, o modelo de classes de COMPOR-F é suficientemente simples para ser implementado em outras linguagens, o que não ocorre com JAF por causa do acoplamento ao modelo *JavaBeans*.

JAFMAS [Cha97]

Resumo

Java-based Agent Framework for Multi-Agent Systems é um arcabouço para representar e desenvolver protocolos e conhecimento de cooperação em SMA's. JAFMAS dá suporte a uma metodologia para desenvolvimento de SMA's. A metodologia considera os seguintes

passos: definição do agente, identificação das conversações, identificação das regras de conversações, análise do modelo de conversação e implementação do sistema.

O arcabouço foi concebido com mais atenção à implementação de protocolos de interação entre agentes e coordenação na comunidade multiagentes.

Arquitetura

A arquitetura multiagentes tem foco sobre a comunicação, lingüística e coordenação dos agentes que fazem parte do sistema. A comunicação suporta difusão(*broadcast*) e comunicação dirigida. A interação é baseada na idéia de conhecimento social [Cos97] e é representada por classes de Conversação, cada conversação funciona em uma *Thread* e é regida por regras de conversação. Um agente é então composto de um modelo social e um modelo de comunicação. Há ainda um modelo local de dados, implementado pelo usuário do arcabouço e um modelo de interface gráfica.

Implementação

JAFMAS foi desenvolvido utilizando a linguagem Java e a comunicação foi implementada utilizando Java/RMI.

Instanciações

Implementação do problema das n-rainhas e do problema da cadeia de suprimentos, onde o sistema deve gerenciar uma rede de fornecedores, fábricas, centros de distribuição e locais de venda, utilizando um SMA, em que cada agente implementa uma função da cadeia e sua coordenação com outros agentes visa um planejamento global de sucesso.

Relação com COMPOR-F

A principal diferença entre JAFMAS e os outros arcabouços é a definição de protocolos de interação via regras de conversação e comunicação. A divisão do agente em módulos (principalmente social e comunicação) torna a arquitetura mais modular e transparece uma maior independência de implementação entre os seus componentes. O arcabouço COMPOR-F encapsula em componentes as funcionalidades de interação, ou seja, um componente que

permita a definição da interação via regras pode ser implementado. A arquitetura de um agente COMPOR-F também é dividida em módulos contextuais:inteligente, social e de distribuição. A abordagem de componentes é o principal diferencial de COMPOR-F em relação a JAFMAS, que é orientado a objetos.

OAA [MCM99]

Resumo

Open Agent Architecture é um arcabouço arquitetural para desenvolvimento de aplicações de natureza distribuída baseada em agentes autônomos. A comunicação e cooperação entre estes agentes é intermediada por facilitadores que têm a funcionalidade de identificar compatibilidades entre os serviços de um agente e as capacidades dos outros.

Arquitetura

A arquitetura de comunicação entre agentes é baseada na arquitetura de quadro-negro. Um repositório global onde agentes publicam seus serviços para que outros agentes os requisitem. A interação entre agentes é feita com base em uma Linguagem de Comunicação Inter-agente (ICL).

Implementação

Existem bibliotecas em diversas linguagens para utilização do arcabouço, dentre elas PROLOG, Java, C++, Visual Basic e Delphi.

Instanciações

O arcabouço OAA foi utilizado para o desenvolvimento de mais de quinze aplicações, integrando tecnologias como reconhecimento de voz, processamento de imagens, colaboração multi-usuário, extração de texto, planejamento e realidade virtual.

Relação com COMPOR-F

O uso da arquitetura de quadro-negro como um padrão de interação entre os agentes é um fator negativo de OAA. Há poucas considerações sobre o projeto do arcabouço, utilização

de padrões de projeto ou pontos de reutilização. O arcabouço COMPOR-F, assim como os outros arcabouços apresentados nesta análise, não determina a forma de interação entre os agentes. O motivo é simples: a interação entre os agentes depende da arquitetura do sistema a ser desenvolvido.

RMIT [KKPC00]

Resumo

O arcabouço RMIT é fortemente baseado em padrões de projeto e é composto de oito sub-arcabouços, que podem ser considerados independentes, um para cada módulo dos agentes. Os comportamentos dos agentes são implementados pelos seguintes módulos:

1. **Módulo sensor**, para percepção do ambiente;
2. **Módulo de crenças**, onde é representado o modelo do próprio agente e o modelo do ambiente;
3. **Módulo de raciocínio**, onde é determinado o que fazer, com base em seus objetivos, planos, capacidades e crenças.
4. **Módulo de ação**, onde são representadas as intenções do agente, ou seja, planos em execução. Planos pendentes compõem uma lista de espera.
5. **Módulo de colaboração**, onde é determinado como ocorre a colaboração entre os agentes.
6. **Módulo de tradução**, onde ocorre a tradução de linguagens ou semânticas (ontologias) entre agentes com dialetos diferentes.
7. **Módulo de mobilidade**, que é utilizado quando agentes estão em diferentes localizações.

Para cada um dos oito módulos acima descritos existe um arcabouço independente baseado em padrões de projeto.

Arquitetura

A arquitetura é baseada nos módulos que compõem o agente. Um agente da sociedade é composto pelos oito módulos dispostos de baixo para cima, na ordem descrita anteriormente. As mensagens são transmitidas entre módulos adjacentes nos dois sentidos, como requisição e resposta a eventos ou solicitação de serviços.

Implementação

A implementação foi feita em Java com forte utilização de padrões de projeto.

Instanciações

A aplicação-exemplo do arcabouço RMIT é um agente de gerenciamento de rede (NMA) que tem um servidor de gerenciamento de rede como objeto co-existente no ambiente. O agente possui crenças que modelam a informação de gerenciamento do servidor.

Relação com COMPOR-F

Assim como COMPOR-F, o arcabouço RMIT é baseado em padrões de projeto e a abordagem de sub-arcabouços permite maior adaptabilidade a mudanças, pois cada uma das características específicas podem ser re-projetadas de forma independente. A definição de um agente como a composição de oito sub-arcabouços atribui uma complexidade ao agente que deveria vir por demanda. Ou seja, considera-se por definição que um agente possui mobilidade, raciocínio, crenças, dentre outras características. Em COMPOR-F não se definem as características de um agente, com exceção da autonomia, sem um domínio de aplicação. De acordo com a aplicação a ser desenvolvida, a demanda por complexidade é traduzida em funcionalidade nos agentes. Esta flexibilidade provida pelo projeto baseado em componentes do arcabouço é uma das principais características de COMPOR-F.

SAF [GSP98]

Resumo

Standard Agent Framework é um arcabouço orientado a objetos extensível que provê meios de construção de SMA's através da especialização de classes-base padrões e composição de classes-componente. Possui duas classes abstratas associadas: agente e agência. Agência é o "habitat" dos agentes. Uma aplicação é então modelada como um grupo de agências que interagem através dos agentes que as compõem.

Arquitetura

Sistemas de agentes padrões são implementados através da especialização de quatro classes: AgênciaPadrão, AgentePadrão, AtorHumano e Recurso. Uma instância da classe Agência-Padrão é composta por coleções de agentes padrões, atores humanos e recursos. As classes de agentes, atores e recursos estão ligadas a uma agência através do seu projeto de classes, utilizando o padrão *AbstractFactory* [GHJV95]. Uma instância de AgentePadrão é composta por coleções de objetos que definem sua estrutura e seu comportamento. Os mecanismos de comunicação e de inferência para alcançar os objetivos de um agente são genéricos e podem ser especializados com base na solução padrão. Atores humanos são objetos no domínio do agente que representam pessoas interagindo com agentes através de algum dispositivo de interface. Instâncias da classe Recurso provêem controle de concorrência e protocolo de acesso aos recursos da agência. A linguagem de comunicação entre os agentes pode ser definida como especialização da linguagem padrão, que no exemplo apresentado em [GSP98] é KQML [FFMM94].

Implementação

A especificação do arcabouço não tem dependência de linguagens. Porém, as aplicações apresentadas em [GSP98] que se basearam no Sistema Padrão de Agentes, que é a instância direta do arcabouço, possui como infra-estrutura a abordagem de objetos distribuídos, protocolos de ciclo de vida de objetos e meta-objetos. Logo, a utilização de uma linguagem que disponibilize API's para tal infra-estrutura seria desejável.

Instanciações

O arcabouço foi utilizado em três grandes projetos: um sistema de comércio eletrônico no domínio de transporte e manufatura; um simulador para pesquisa de robôs; e uma máquina de busca para a WWW.

Relação com COMPOR-F

A definição de um sistema padrão de agentes servindo de modelo para a implementação de SMA's é uma forma de diminuir o trabalho do projetista da aplicação, provendo funcionalidades comuns a SMA's. Este modelo padrão não é provido por COMPOR-F, mas é uma das perspectivas futuras do ambiente COMPOR-E descrito no Capítulo 6. Assim como *Brainstorm/J*, utiliza a abordagem de meta-objetos. Sendo assim, os agentes são considerados objetos com características de inteligência e autonomia. Em COMPOR-F, as características e funcionalidades dos agentes são encapsuladas nos componentes, os quais são implementados utilizando Java. Se o agente possui "componentes inteligentes", o agente é inteligente. Em SAF é necessário implementar estas características com os meta-objetos, não se valendo assim da possibilidade de reutilização das funcionalidades.

Na Tabela C.1 é apresentado o resultado da análise comparativa dos arcabouços. Os parâmetros são: a utilização de componentes e padrões no projeto do arcabouço; a disponibilização de uma infra-estrutura de comunicação; a definição de um módulo, camada ou componente que possa prover características inteligentes ao agente (aprendizado ou inferência de conhecimento, por exemplo) ou integração com outros arcabouços que implementem tais características; e a utilização de padrões de linguagens e protocolos de interação como FIPA ou KQML para a interação entre os agentes.

Arcabouços	**Característica**				
—	Componentes	Padrões	Infra-estrutura	Inteligência	FIPA/KQML
Brainstorm/J		●	●		●
DECAF			●		●
DSAP		●	●	●	
FraMAS		●	●	●	●
JACK			●		●
JADE			●	●	●
JAF	●		●		●
JAFMAS			●		●
OAA			●		
RMIT		●	●		●
SAF		●	●		●

Tabela C.1: Tabela comparativa dos arcabouços para a construção de sistemas multiagentes

Apêndice D

Análise comparativa das ferramentas e ambientes correlatos

Algumas ferramentas para o auxílio no desenvolvimento de aplicações multiagentes foram propostas e são analisadas com base nos critérios enumerados a seguir.

1. **Fases do desenvolvimento de software apoiadas pela ferramenta:** que fases do desenvolvimento de software são contempladas pela ferramenta (análise, projeto, implementação, testes, gerência etc)? A ferramenta está baseada em alguma metodologia para o desenvolvimento de sistemas multiagentes?

2. **Infra-estrutura:** qual a infra-estrutura provida pela ferramenta em relação a protocolos de interação e tecnologias de comunicação?

3. **Escalabilidade:** existe a possibilidade de acréscimo de novos módulos à ferramenta? Esta funcionalidade é gerenciada pela própria ferramenta, ou seja, via interface gráfica é possível adicionar novas funcionalidades (*plugins*)?

4. **Suporte à internacionalização:** a ferramenta provê suporte à internacionalização? [Gre03]

5. **Relação com COMPOR-E:** Relação entre a ferramenta e o ambiente COMPOR-E.

Nenhuma das ferramentas analisadas leva em conta a escalabilidade e a internacionalização, ao menos nas referências pesquisadas. Os outros tópicos de análise são apresentados a seguir.

agentTool [Lab03]

Fases do desenvolvimento de software

A ferramenta agentTool dá suporte à análise e projeto, permitindo a construção de diagramas da metodologia MaSE [DWS01]. Em relação à implementação, gera um "esqueleto" do agente com base nos diagramas criados na fase de análise, apenas com a assinatura de alguns métodos. Além disso, permite a execução de testes, embora sejam apenas para a verificação de erros nas interações entre os agentes.

Infra-estrutura

A partir da definição bem detalhada na análise e no projeto, pode-se gerar o código do agente. agentTool disponibiliza quatro geradores de código, que têm como entrada os diagramas definidos através da ferramenta. Um editor de ontologias também é disponibilizado a partir da versão 2.0.

Relação com COMPOR-E

Apesar do suporte provido por agentTool em relação à construção dos diagramas de análise e projeto, o código gerado a partir dos diagramas é aparentemente restrito ao esqueleto das classes de agentes. Não há suporte a funcionalidades como inferência de conhecimento e aprendizagem, nem fica explícito como acoplar estas funcionalidades à ferramenta. O ambiente COMPOR-E não dá suporte a estas características, mas permite que sejam construídos componentes que as implemente. Esta escalabilidade não é definida nas outras ferramentas desta análise e é uma das principais contribuições de COMPOR-E.

FIPA-OS [Emo03]

Fases do desenvolvimento de software

FIPA-OS é uma ferramenta para a construção de agentes de acordo com a arquitetura definida por FIPA. A ferramenta dá suporte apenas à implementação e à gerência da execução dos

agentes.

Infra-estrutura

Além da infra-estrutura de comunicação via FIPA-ACL, FIPA-OS também fornece alguns componentes adicionais, como um componente que provê integração com o JESS [FH02]. Outra característica deste ambiente é a possibilidade de visualização das tarefas que estão sendo executadas pelos agentes.

Relação com COMPOR-E

Apesar de FIPA-OS ser utilizada como um ferramenta de desenvolvimento de sistemas multiagentes, ela apenas dá suporte à interação entre os agentes. COMPOR-E é um ambiente que aborda várias fases de desenvolvimento, não se restringindo apenas a uma das características dos SMA's. As funcionalidades providas por FIPA-OS estão contidas nas ferramentas disponibilizadas por COMPOR-E.

JADE [BRP99]

Fases do desenvolvimento de software

JADE dá suporte apenas à execução e gerenciamento dos agentes.

Infra-estrutura

Assim como FIPA-OS, JADE disponibiliza uma API para a definição da interação entre os agentes e uma ferramenta para o gerenciamento dos agentes da sociedade. A infra-estrutura de JADE é provida pelo arcabouço JADE, o qual foi descrito no Apêndice C.

Relação com COMPOR-E

JADE e FIPA-OS são bem semelhantes. Ambos têm como principal contribuição a implementação dos padrões FIPA e dão suporte à implementação e gerência da execução do

sistema multiagentes. As mesmas considerações feitas em relação a FIPA-OS, também são válidas para JADE.

JATLite [JPC00]

Fases do desenvolvimento de software

Nenhuma consideração sobre fases de desenvolvimento é feita nas referências pesquisadas. JATLite representa apenas um conjunto de classes Java para implementar a interação entre agentes.

Infra-estrutura

Possui suporte para KQML e pode ser adaptado para contemplar FIPA-ACL.

Relação com COMPOR-E

JATLite é uma das primeiras ferramentas para a construção de SMA's. Fornece suporte à interação entre os agentes mas suas funcionalidades estão muito aquém daquelas disponibilizadas por JADE e FIPA-OS.

MADKit [GFM03]

Fases do desenvolvimento de software

Nenhuma consideração sobre fases de análise e projeto é feita nas referências pesquisadas. A ferramenta dá suporte à execução e gerenciamento dos agentes contidos em um repositório.

Infra-estrutura

Define classes para as mensagens a serem trocadas entre os agentes. Possui classes KQMLMessage e ACLMessage para a comunicação via KQML e FIPA-ACL, respectivamente. Assim como a maioria das ferramentas, MADKit provê uma API java para que o desenvolvedor possa estender uma classe *Agent* onde está determinado o ciclo de vida dos agentes. O

comportamento dos agentes é definido através da implementação dos métodos definidos na classe *Agent*, tais como *activate*,*live* e *end*.

Relação com COMPOR-E

Em MadKIT, os agentes podem ser codificados em Java, Scheme, Jess, Python ou BeanShell [GFM03]. A ferramenta em si é apenas um repositório de agentes. A partir de um XML, pode-se associar agentes a ícones e agrupá-los em uma barra de ícones que é exibida na ferramenta. Desta forma, ao contrário de COMPOR-E, o desenvolvimento dos agentes não é gerenciado pela ferramenta. MadKIT apenas os exibe como componentes de interface gráfica e gerencia a sua execução. Uma vantagem em relação ao COMPOR-E é que os agentes podem ser implementados em diferentes linguagens.

RETSINA [Ret03]

Fases do desenvolvimento de software

RETSINA não está vinculada a metodologias de desenvolvimento, provê apenas suporte à execução e gerenciamento dos agentes.

Infra-estrutura

Utiliza KQML como linguagem de comunicação entre agentes. Nenhuma consideração é feita em relação à escalabilidade da infra-estrutura em termos de tecnologias e linguagens de comunicação.

Relação com COMPOR-E

Assim como as outras ferramentas, RETSINA não dá suporte ao acoplamento de novas funcionalidades aos agentes. Uma vez que os agentes não são definidos com base na componentização das funcionalidades. Esta é a principal diferença entre COMPOR-E e as demais ferramentas.

Zeus [Zeu03]

Fases do desenvolvimento de software

Zeus não está vinculada a metodologias de desenvolvimento. Provê as funcionalidades de execução dos agentes e gerenciamento das interações.

Infra-estrutura

Zeus disponibiliza a implementação da especificação de FIPA-ACL em sua biblioteca de classes. Nenhuma consideração é feita quanto à possibilidade de modificação ou adição de tecnologias de comunicação e novas linguagens de interação.

Relação com COMPOR-E

Zeus está vinculado à especificação de tarefas para cada agente criado. Apesar da ferramenta ser utilizada para o desenvolvimento baseado em agentes, não ficam claras como seria sua utilização em um sistema de larga escala. Os exemplos utilizados são muito simples se comparados à complexidade de um sistema multiagentes. O estudo de caso de COMPOR-E é uma aplicação inerentemente complexa no domínio de empresas virtuais. O foco de COMPOR-E está no desenvolvimento em larga escala.

Na Tabela D.1 é apresentado o resultado da análise comparativa das ferramentas. Os parâmetros são a infra-estrutura disponibilizada e as fases de desenvolvimento representadas por Análise(A), Projeto(P), Implementação(I), Testes(T) e Gerência de execução(G).

Ferramentas	**Características**	
—	Fases	Infra-estrutura
agentTool	(A)(P)(I)(T)(G)	Construção de diagramas/Comunicação
FIPA-OS	(I)(G)	Comunicação - FIPA/ACL
JADE	(I)(G)	Comunicação - FIPA/ACL
JATLite	(I)	Comunicação - FIPA/ACL e KQML
MADKit	(I)(G)	Comunicação - FIPA/ACL
RETSINA	(I)(G)	Comunicação - KQML
Zeus	(I)(G)	Comunicação - FIPA/ACL

Tabela D.1: Tabela comparativa das ferramentas para a construção de sistemas multiagentes