UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

DESENVOLVIMENTO DE UMA PLATAFORMA MULTIAGENTE COGNITIVA PARA BANCOS DE DADOS AUTÔNOMOS: O CASO DO DBSITTER PLUS

TRABALHO DE GRADUAÇÃO

Aluno: Davi Lyuma Anabuki (dla@cin.ufpe.br)
Orientadora: Patrícia Cabral de Azevedo Restelli Tedesco (pcart@cin.ufpe.br)

Setembro de 2006

Resumo

Este Trabalho de Graduação se propõe a fazer alterações na arquitetura do DBSitter, um sistema multiagente (SMA) desenvolvido para auxiliar os administradores de banco de dados no gerenciamento e monitoração de Sistemas de Gerenciamento de Banco de Dados (SGBD), de modo a deixá-lo de acordo com os padrões da FIPA - Foundation for Intelligent Physical Agents. As alterações realizadas têm o intuito de tornar o sistema mais interoperável e extensível, permitindo a comunicação com agentes que implementem os padrões da FIPA com o mínimo de alterações.

Agradecimentos

Agradeço primeiramente à minha família, por me dar o suporte necessário para que pudesse chegar aonde estou.

Agradeço também ao amigo Paulo Maciel, que me ajudou durante esta caminhada.

Agradeço à professora Patrícia, por todo o auxílio, paciência e confiança, indispensáveis, durante este trajeto.

Índice

1. Introdução

A indústria de Tecnologia da Informação (TI) tem passado por um desenvolvimento tecnológico acentuado, com um aumento do poder computacional disponível para os usuários crescendo a uma taxa quase exponencial, o que vem a confirmar as previsões de Moore [44]. Esse crescimento tem possibilitado a automação cada vez maior de tarefas, porém, tem gerado como subproduto um aumento da complexidade dos sistemas, criando uma demanda cada vez maior por profissionais especializado para manutenção dos mesmos.

Horn [4] cita como uma alternativa para esta demanda a criação de sistemas que se auto-gerenciem. Uma das maneiras de atingir este objetivo é construir sistemas autônomos, que inicialmente poderiam ser aplicados a domínios específicos. Nesse sentido, uma área que traz grandes desafios é a de Banco de Dados. Sendo alguns deles: (1) o ambiente cada vez mais complexo, devido à grande quantidade de variáveis que, em conjunto, afetam o desempenho do Sistema Gerenciador de Banco de Dados (SGBD); (2) o tratamento de grandes volumes de informação e (3) necessidade de monitoração constante.

Esse problema foi abordado por Carneiro [1] através da formulação do DBSitter, um sistema multiagente (SMA) para monitoração e gerenciamento de SGBD. Este auxilia o Administrador de Banco de Dados (ABD) no processo de manutenção do SGBD, atuando ao encontrar um problema, seja através da sugestão de uma solução ao ABD ou da aplicação automática de uma solução adequada.

Este trabalho se propõe a alterar o DBSitter de modo que ele esteja de acordo com a padronização adotada pela FIPA para SMA, utilizando a metodologia de desenvolvimento SMA Gaia durante este processo.

1.1 Objetivo

Este documento tem visa apresentar os resultados obtidos da pesquisa realizada durante a execução deste trabalho de graduação, que teve os seguintes objetivos:

1. Reavaliação do problema do DBSitter [1] sob a ótica de uma metodologia de desenvolvimento orientado a agentes;
2. Adequação às normas de SMA sugeridos pela FIPA [13];
3. Prova de conceito através da implementação das mudanças sugeridas utilizando o framework de desenvolvimento de SMA JADE [38].

1.2 Metodologia

Para a realização deste trabalho foi realizado um estudo do problema abordado pelo DBSitter, o que incluiu a geração de um documento de requisitos e a geração do diagrama i* [42] do mesmo. Estudamos algumas das metodologias de desenvolvimento multiagente existentes na atualidade, analisando suas características, e escolhendo uma destas para aplicação no problema estudado.

Fizemos também um estudo da padronização sugerida pela FIPA para SMA e implementamos novos agentes para o DBSitter, de acordo com os padrões que consideramos relevantes, fazendo uso do framework JADE.

1.3 Organização do Documento

Este documento está organizado da seguinte maneira:

Capítulo 2 – Apresenta alguns conceitos básicos, como a noção de computação autônoma e conceitos de SMA, assim como a relação entre eles. Fazemos também uma breve discussão de algumas metodologias de desenvolvimento de SMA e uma descrição dos padrões adotados pela FIPA relevantes ao nosso estudo.

Capítulo 3 – Apresenta a tecnologia utilizada durante a implementação dos agentes do nosso trabalho, o framework Jade.

Capítulo 4 – Apresenta o nosso trabalho, mostrando o que foi realizado e os resultados obtidos.

Capítulo 5 – Apresenta nossas conclusões e sugestões para trabalhos futuros.

2. Conceitos Básicos

Este capítulo tem como objetivo abordar os conceitos básicos diretamente relacionados ao nosso trabalho. Discutiremos um pouco sobre os conceitos de computação autônoma e SMA e faremos uma breve discussão sobre o relacionamento entre eles e de alguns conceitos sobre computação autônoma relacionada a bancos de dados.

2.1. Computação Autônoma

Sistemas computacionais têm aumentado o seu poder de processamento em uma escala quase exponencial. Os programas têm sido gerados de maneira a fazer uso desta capacidad, automatizando cada vez mais tarefas. Entretanto, isso tem gerado o aumento de complexidade dos sistemas, que necessitam cada vez mais de mão de obra especializada para realizar sua gerência e manutenção.

De acordo com Horn em seu manifesto de computação autônoma [4], a indústria de TI está enfrentando este problema, da *complexidade*, e uma das maneiras encontradas para combatê-lo é através da utilização dos conceitos de *computação autônoma*.

O princípio da computação autônoma é baseado no funcionamento do sistema nervoso autônomo humano, que controla as funções básicas do organismo (e.g.: a respiração e a freqüência dos movimentos cardíacos), permitindo que o cérebro possa se concentrar em funções de alto-nível (e.g.: o destino de nossa caminhada). De maneira similar, os sistemas computacionais autônomos se propõem a permitir que o usuário possa se concentrar em funções de alto-nível, enquanto o sistema se encarrega de lidar automaticamente com as funções básicas.

O problema da automatização do gerenciamento de recursos computacionais não é um problema novo [5]. Há décadas os componentes de sistemas e o software utilizado

têm evoluído para lidar com o aumento da complexidade do controle do sistema, do compartilhamento de recursos e do gerenciamento operacional. A computação autônoma é considerada o próximo passo para lidar com os ambientes distribuídos e complexos de hoje.

Uma amostra da importância dessa abordagem pode ser vista através do enfoque recente da IBM, com um esforço que pôde ser observado através da publicação do manifesto de computação autônoma, onde são identificados os pontos chaves de um sistema autonômico, e tenta atrair tanto a academia quanto a indústria a participarem deste esforço.

O manifesto de computação autônoma identifica oito elementos chaves que devem estar contidos em um sistema autônomo:

1. O sistema deve se conhecer;
2. O sistema deve se configurar e reconfigurar sob diferentes condições;
3. O sistema deve procurar maneiras de otimizar o seu funcionamento;
4. O sistema deve realizar algo além da recuperação, ele deve estar preparado para se recuperar de eventos rotineiros e extraordinários;
5. O sistema deve ser um especialista em auto-proteção;
6. O sistema deve conhecer o ambiente e o contexto, agindo de acordo com os mesmos;
7. O sistema não pode existir em um ambiente hermético;
8. O sistema deve antecipar os recursos otimizados necessários, enquanto mantêm sua complexidade escondida.

Em termos mais práticos, foi identificado por Kephart e Chess [3] que um sistema autônomo tem como objetivo o auto-gerenciamento, que pode ser obtido através de quatro aspectos, sendo eles: (1) auto-configuração, (2) auto-otimização, (3) auto-recuperação e (4) auto-proteção. Eles também identificam que um elemento autônomo típico, consiste em um ou mais elementos gerenciados em conjunto por um gerente autônomo que os controla e representa. As características desses elementos autônomos, que são autonomia, proatividade e interação baseada em objetivos são características de agentes inteligentes [3] , o que indica a importância que os conceitos de arquitetura orientada a agentes podem ter em computação autônoma.

2.2. Sistemas Multiagente

Sistemas multiagente são, de acordo com Weiss [40], sistemas nos quais vários agentes inteligentes interagem, em busca de um conjunto de objetivos ou executam algum conjunto de tarefas. Nesse contexto, *agentes* são entidades computacionais, autônomas, que percebem o ambiente através de sensores e atuam sobre ele através de atuadores. *Inteligentes* indica que os agentes perseguem seus objetivos e executam tarefas que visam a otimização de alguma medida de performance. E *interagem* indica que os agentes podem ser influenciados por outros agentes ou talvez por humanos na busca pelos seus objetivos e durante a execução de suas tarefas.

Wooldridge [41] define agente de uma maneira mais refinada, indicando as seguintes características de um agente: (1) são entidades de resolução de problemas claramente identificáveis, com limites e interfaces bem-definidas; (2) são situados (embutidos) em um ambiente particular – eles recebem entradas de acordo com o estado de seus ambientes através de sensores, e atuam no ambiente através de atuadores; (3) são projetados de modo a executar um objetivo específico – eles tem objetivos particulares para atingir; (4) são autônomos – ou seja, eles têm controle tanto do seus estados internos como de seu próprio comportamento; (5) são capazes de exibir um comportamento de resolução de problemas flexível, na busca pelos seus objetivos de projeto. Eles precisam ser tanto reativos (podem responder em tempo hábil às mudanças que ocorrem no seu ambiente) como proativos (possíveis de atuar antecipadamente de acordo com objetivos futuros).

Tanto Wooldridge [41] quanto Weiss [40] concordam que a área de engenharia de sistemas orientados a agentes ainda é uma área que não possui um conceito único e universal amplamente aceito para agentes. Ou seja, eles tentam se concentrar em características que sejam necessárias, mas não suficientes, de modo a poder descartar sistemas que não sejam claramente orientados a agentes, e poder classificar uma ampla gama de SMA.

Franklin e Graesser, em [39], fazem uma avaliação de vários sistemas orientados a agentes e chegam à seguinte formalização para agentes:

"Um agente autônomo é um sistema situado dentro de um ambiente, e faz parte dele, que consegue perceber este e atuar sobre ele, com o passar do tempo, em busca de seus próprios objetivos, e de tal modo que ele influencie o que vai perceber no futuro."

Já Russell e Norvig, em [43] fazem a seguinte colocação:

"A noção de um agente foi concebida para ser uma ferramenta para analisar sistemas, não para ser uma caracterização absoluta que divida o mundo em agentes e não-agentes."

A pesquisa em torno de SMA tem crescido bastante nos últimos anos, em especial na área de Engenharia de Software, na busca de uma metodologia que torne possível a adoção desta tecnologia em massa. Weiss [40] define como razões para o crescente interesse em SMA: (1) as necessidades tecnológicas e de aplicação, pois SMA oferecem uma maneira de entender, gerenciar e usar sistemas distribuídos de larga escala, dinâmicos, abertos e sem distribuição; (2) a visão natural de sistemas inteligentes oferecida por SMA.

Além disso, SMA têm a capacidade de oferecer as seguintes propriedades:

- Aumento de velocidade e eficiência;
- Aumento de robustez e confiança;
- Aumento de escalabilidade e flexibilidade;
- Diminuição de custos;
- Aumento na velocidade de desenvolvimento e reuso.

Jennings, por sua vez, em [41], cita como características importantes do desenvolvimento de SMA: (1) a sua capacidade de decomposição de problemas, (2) a sua capacidade de abstração, e (3) a capacidade de lidar com organizações.

Mas, qual a razão de se utilizar SMA? E quais as suas aplicações? Como Jennings descreve em [41], a utilização de SMA deriva da necessidade de alternativas para o desenvolvimento de sistemas com um grau de complexidade cada vez mais alto, e o paradigma de orientação a agentes, fornece mecanismos para decomposição, abstração e organização.

A utilização de SMA sugere a decomposição do problema em termos de agentes autônomos, gerando uma representação natural de sistemas complexos que são invariavelmente distribuídos e que invariavelmente tem múltiplos focos de controle.

Em termos de abstração, SMA oferecem uma abstração que auxilia a diminuição da distância semântica entre as unidades de análise utilizadas e o problema, existindo uma correlação clara entre as noções de subsistemas e as organizações de agentes.

Em termos de organização, SMA possuem uma representação explícita de relacionamentos organizacionais e de estrutura, além de ter mecanismos computacionais concomitantes para a formação, manutenção e remoção de organizações. Esse poder de representação permite uma maior adequação a sistemas complexos.

Huhns [40] cita como outras razões para estudar SMA:

- Problemas utilizando SMAS são às vezes mais fáceis de entender e mais fáceis de desenvolver, em especial quando o problema é distribuído por si só. A distribuição pode levar a algoritmos que não seriam descobertos através de uma solução centralizada.
- Algumas vezes uma solução centralizada é impossível, quando os atores (organizações) precisam manter as informações o mais privado possível.
- A informação envolvida é necessariamente distribuída e está em sistemas de informação grandes e complexos nos seguintes sentidos:
    1. São geograficamente distribuídos;
    2. Podem ter vários componentes;
    3. Podem ter um conteúdo muito grande;
    4. Podem ter um escopo muito grande.

De acordo com isso, podemos dizer que SMA podem ter um papel importante no desenvolvimento futuro de sistemas computacionais. Além dessas características, e essas motivações, é importante também salientar que SMA não são simplesmente sistemas distribuídos ou concorrentes. Eles possuem características próprias [18], como a autonomia de cada agente, assim como a existência de interações de cunho econômico entre os agentes, que os diferenciam de sistemas puramente distribuídos. Sendo que eles

também não são simplesmente Inteligência Artificial (IA), pois existe a importância do aspecto social, assim como não são somente teoria dos jogos ou a aplicação de teorias sociais, mas um misto de ambos.

2.3. Computação Autônoma, Sistemas Multiagente e SGBD

Características que proporcionem algum grau de independência da intervenção humana têm aumentado versão após versão e historicamente os principais SGBD de mercado já fizeram avanços na área de automatização de tarefas desde o final da década de 1970, mas impulsionados pelas preocupações dos laboratórios de pesquisa da IBM e pela força da concorrência de mercado que isso gerou, apenas recentemente verdadeiras inovações vêm sendo lançadas como características de autonomia. Esses esforços proporcionariam a criação de um novo conceito, o SGBDA – Sistema de Gerenciamento de Bancos de Dados Autônomo.

Duas iniciativas de pesquisa específicas na linha de autonomia em SGBD vêm sendo conduzidas por dois dos maiores fabricantes de software: o projeto SMART (*Self-Managing and Resource Tuning*) da IBM para o Banco de Dados DB2 e o projeto AutoAdmin da Microsoft.

Entretanto, estas duas iniciativas de pesquisa não têm um foco específico no desenvolvimento de SGBDA baseado em agentes. Um esforço específico nesse sentido pode ser visto no trabalho desenvolvido por Bigus et al [45], Diao et al [46] e Costa e Lifschitz [48] no desenvolvimento de sintonia para performance. Assim como pode ser percebido em Carneiro [1, 2].

Lifschitz e Macedo [12] propõem uma classificação das arquiteturas de integração entre agentes e SGBD, sendo elas: (1) em camadas (*layered*); (2) integrada (*integrated*) e (3) embutida (*built-in*), conforme representadas na figura 2.1.

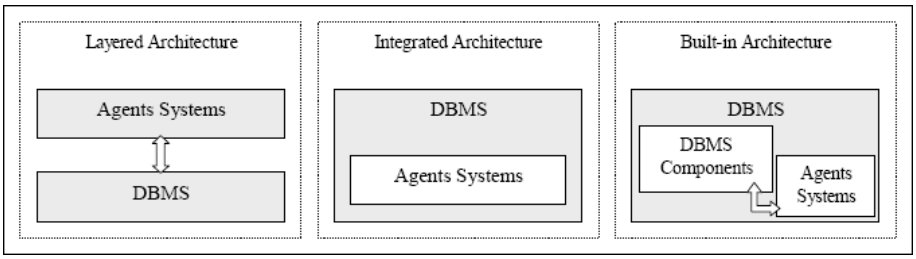

Figura 2.1 – *Arquiteturas para integração de sistemas multiagente e SGBD - adaptado de [12]*

A arquitetura em camadas representa a implementação de um sistema de agentes que trabalha sobre o SGBD, sem a necessidade de alterá-lo. Nesta arquitetura, o agente funciona de maneira similar a um mediador, que recebe, manipula e traduz as requisições ao SGBD.

A arquitetura integrada é baseada na substituição dos componentes tradicionais de um SGBD por subsistemas de agentes, onde estes subsistemas precisam apresentar todas as funcionalidades dos componentes originais do SGBD.

A arquitetura "built-in", considera o uso de um SGBD existente, onde os agentes passam a interceptar chamadas do código do banco, alterando-as. Nesse caso, os agentes estão embutidos dentro do SGBD, usando os serviços existentes fornecidos pelos componentes do SGBD, enquanto incorporam novas funcionalidade.

Dentro desta classificação, tanto o DBSitter quanto o DBSitter Plus implementam uma arquitetura baseada em camadas.

2.4 Metodologias de Desenvolvimento para Sistemas Multiagente

Na seção anterior, comentamos sobre a necessidade da existência de um esforço na área de Engenharia de Software, para que a adoção do paradigma orientado a agentes pudesse acontecer. Nesta seção, iremos descrever alguns resultados dos esforços das pesquisas existentes, refletidas nas metodologias de desenvolvimento.

De acordo com [17], não existe uma metodologia “melhor” do que outra, mas simplesmente metodologias mais adequadas para cada caso. Podemos citar como exemplo, a existência de metodologias derivadas da extensão de diagramas de UML [31, 32] que podem ser adequadas ao desenvolvimento de sistemas em equipes que tenham experiência no desenvolvimento com uso de UML, o que pode facilitar o processo de transição. Existem metodologias que se baseiam na visão dos agentes como uma comunidade, dando uma ênfase maior à comunicação entre os agentes [17], enquanto outras atuam de maneira a resolver os problemas com foco nos agentes individuais, sendo o comportamento do grupo uma conseqüência do comportamento dos agentes individualmente [34, 35].

2.4.1 Estudo de metodologias

Para a escolha da metodologia a ser utilizada neste trabalho, fizemos um breve estudo das metodologias disponíveis. Optamos pela metodologia Gaia devido à sua abordagem top-down, que vislumbra o processo trabalhando com uma abordagem inicialmente voltada para a organização, em contraste com metodologias como MaSE (Multiagente Systems Engineering) [36], cujo comportamento da sociedade de agentes é uma conseqüência do comportamento individual de cada componente.

Ela também foi escolhida em detrimento de metodologias que estendem UML, como MAS-ML ou aUML, devido ao fato das mesmas não possuírem o mesmo poder expressivo em termos de organização como Gaia.

Outro fato relevante na escolha de Gaia foi a metáfora organizacional adotada pela metodologia, que consideramos se adequar melhor à modelagem do problema enfrentado pelo DBSitter.

2.4.2 A metodologia Gaia

Gaia é uma metodologia que tenta definir um método completo e genérico especificamente moldado para a análise e o projeto de SMA. Ela dá suporte tanto ao nível da estrutura do agente individual como da sociedade de agentes no desenvolvimento de SMA. De acordo com Gaia, SMA são vistos como um número de agentes autônomos

interativos que vivem em uma sociedade organizada onde cada agente representa um ou mais papéis específicos. Sendo a estrutura do SMA definida em termos de um modelo de papéis. O modelo identifica os papéis que os agentes devem representar dentro do SMA e os protocolos de interação entre os diferentes papéis.

A Metodologia Gaia tem seu processo dividido em cinco fases distintas, como pode ser observado na figura 2.2. A fase de análise, a fase de projeto de arquitetura e a fase de projeto detalhado.

A fase de análise tem como objetivo coletar e organizar as especificações que forma a base para o projeto da organização computacional. Essa fase inclui:

- A identificação dos objetivos das organizações que constituem o sistema como um todo e seu comportamento global esperado;
- O modelo de ambiente
- O modelo de papéis preliminar
- O modelo de interações preliminar;
- As regras que a organização deve respeitar e enfatizar no seu comportamento global.

O resultado da fase de análise consiste em um modelo de ambiente, um modelo de papéis preliminar, um modelo de interações preliminar e um conjunto de regras organizacionais, que são utilizadas na fase de projeto.

A fase de projeto de arquitetura, por sua vez, inclui:

- A definição da estrutura organizacional do sistema em termos de sua topologia e regime de controle;
- A derivação de padrões organizacionais;
- A finalização dos modelos preliminares de papéis e de interação.

Uma vez definidos estes artefatos, a fase de projeto detalhado pode ser iniciada, que engloba:

- A definição do modelo de agentes;
- A definição do modelo de serviços.

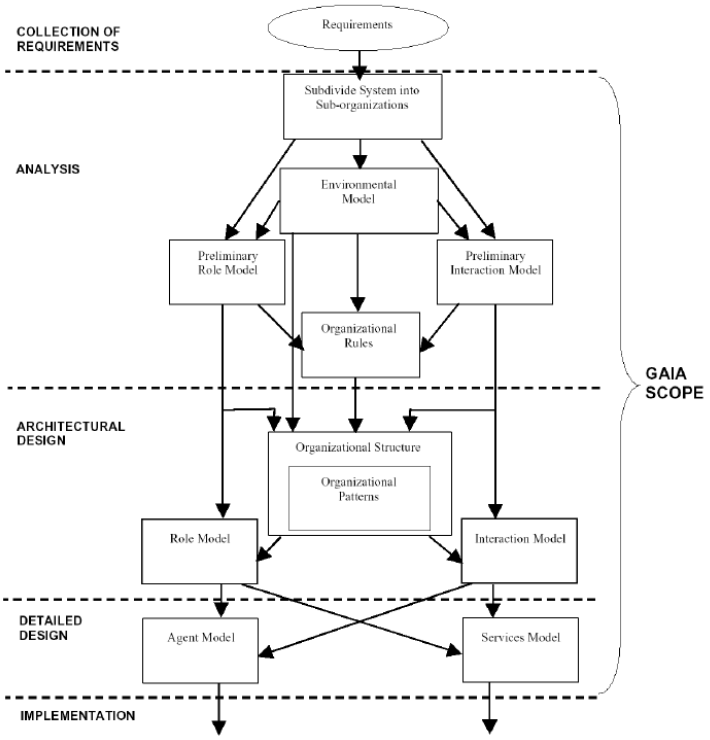

Figura 2.2 – *Modelos da metodologia Gaia e seus relacionamentos no processo Gaia – adaptado de [17]*

2.5 Os padrões da FIPA

A FIPA [1] – Foundation for Intelligent Physical Agents, é uma organização internacional, cujo objetivo é promover o desenvolvimento da indústria de agentes inteligentes através da publicação de especificações abertas de desenvolvimento, de modo a favorecer a interoperabilidade entre agentes e sistemas baseados em agentes.

Durante o nosso estudo, consideramos relevantes os seguintes padrões publicados pela FIPA:

1. FIPA Abstract Architecture Specification – A especificação do padrão de arquitetura abstrata da FIPA;

2. FIPA Agent Communication Language (ACL) Message Structure Specification – A especificação da estrutura de mensagens da linguagem de comunicação entre

agentes;

3. FIPA Communicative Act Library Specification – A especificação de atos de comunicação.

Nós consideramos estes três padrões importantes, pelo seguintes motivos: o padrão (1) define uma arquitetura multiagente que inclui elementos como um serviço de páginas brancas e um serviço de páginas amarelas, que facilitam a descoberta de serviços e de agentes, além de definirem os elementos básicos de um sistema, de modo que seja o mais simples e interoperável quanto possível. O padrão (2), por sua vez, define como deve ser uma estrutura de mensagens para os agentes, de modo que agentes implementados em plataformas diferentes possam se comunicar. E o padrão (3) define o tipo de comunicação que pode ser feita pelos agentes, de acordo com a teoria de atos de fala [40].

2.5.1 A arquitetura abstrata da FIPA

A especificação de arquitetura da FIPA tem como objetivo promover o aumento da interoperabilidade e do reuso. Para atingir esse objetivo, ela busca identificar os elementos fundamentais que precisam estar presente nos SMA e os relacionamentos entre eles, de modo a deixar claro como torná-los interoperáveis.

Podemos citar como elementos importantes da especificação de arquitetura abstrata da FIPA os seguintes elementos:

- O diretório de agentes – também conhecido como o serviço de páginas brancas, que permite a identificação e comunicação com agentes individualmente;
- O diretório de serviços – também conhecido como o serviço de páginas amarelas, que permite a localização de agentes que possuam a capacidade de realizar determinadas tarefas;
- A existência de uma camada de transporte de mensagens – de modo a permitir a comunicação entre os agentes;
- A existência de uma linguagem de comunicação entre agentes – de modo que os

agentes possam compreender as mensagens trocadas.

Um exemplo da transição do modelo abstrato para uma implementação concreta pode ser visualizada na figura 2.3.

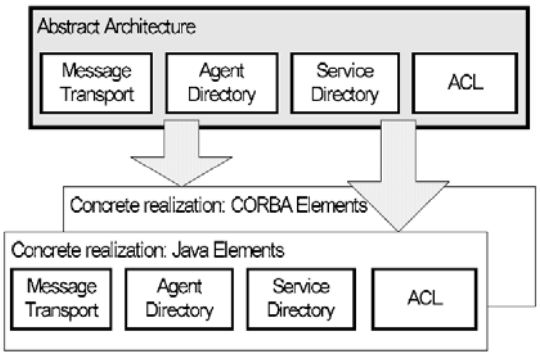

Figura 2.3 – *Arquitetura abstrata mapeada para várias implementações concretas – adaptado de [25]*

2.5.2 A especificação da estrutura de mensagens

Uma mensagem FIPA ACL contém um conjunto de um ou mais parâmetros de mensagens. Precisamente quais parâmetros são necessários para uma comunicação efetiva entre agentes irá variar de acordo com a situação; o único parâmetro mandatório em todas as mensagens ACL é o *performative*, que indica qual a performativa que está sendo enviada. Apesar de que a maioria das mensagens ACL também irá conter os campos *sender*, indicando o remetente da mensagem, *receiver*, indicando o destinatário da mensagem, e *content*, que é o próprio conteúdo da mensagem.

O conjunto completo de parâmetros das mensagens FIPA ACL pode ser visto na tabela 2.1. As implementações têm liberdade para incluir parâmetros além desses especificados, com a condição que esses parâmetros tenham seus nomes precedidos do prefixo “X-“.

Parâmetro	***Categoria do parâmetro***
Performative	Tipo de ato de comunicação
Sender	Participante da comunicação
Receiver	Participante da comunicação
Reply-to	Participante da comunicação
Content	Conteúdo da mensagem
Language	Descrição do conteúdo
Encoding	Descrição do conteúdo
Ontology	Descrição do conteúdo
Protocol	Controle de conversação
Conversation-id	Controle de conversação
Reply-with	Controle de conversação
In-reply-to	Controle de conversação
Reply-by	Controle de conversação

Tabela 2.1 – *Parâmetros de mensagens da FIPA ACL – adaptado de [24]*

2.5.3 A especificação da biblioteca de atos de comunicação

Esta especificação tem como objetivo descrever uma definição formal da linguagem de comunicação e seu significado. Enquanto a especificação da estrutura de mensagem tem como objetivo padronizar a forma das mensagens, a especificação da biblioteca de atos de comunicação tem como objetivo padronizar o significado delas. Ou seja, sua intenção é de fornecer um ponto de referência claro e não-ambíguo para a padronização do significado dos atos comunicativos expressos através de mensagens e protocolos.

2.6 Conclusões

SMA ainda é um campo de pesquisa com vários pontos em aberto. Entretanto, têm se mostrado uma área bastante promissora, exibindo características importantes no desenvolvimento de sistemas. A possibilidade de uma maior abstração e sua visão quase natural de sistemas, decompondo-os em unidades de processamento menores são exemplos que ressaltam a sua importância. Outra vantagem apresentada é a maneira com que este paradigma trata os conceitos de organização, em especial devido ao fato

dos sistemas estarem se tornando cada vez mais conectados e sistemas distribuídos estarem se tornando um padrão nas empresas.

Para que haja esta aceitação, se torna necessário um desenvolvimento muito forte na área de Engenharia de Software para este paradigma, uma preocupação que pode ser percebida através do estudo, desenvolvimento e pesquisa de várias metodologias para SMA, que visam preencher esta lacuna. As metodologias abordadas neste trabalho são apenas algumas de um espectro muito maior de metodologias existentes e em pesquisa.

Por prover uma metáfora organizacional, a metodologia Gaia provê uma forma quase natural de se pensar em SMA, pois podemos pensar nos agentes como uma organização. As fases de análise, projeto de arquitetura e projeto detalhado possuem artefatos bem definidos que permitem a transição entre elas sem maiores dificuldades. Entretanto, a fase de elicitação de requisitos é uma fase importante no desenvolvimento de qualquer sistema, multiagente ou não, e pode ser feita de várias maneiras diferentes. Gaia peca por não definir mecanismos para esta parte do processo.

No nosso desenvolvimento, também percebemos um problema devido à ausência de uma notação adequada para registrar os artefatos utilizados durante o processo de aplicação da metodologia. Apesar dos artefatos sugeridos oferecerem informações suficientes para a aplicação da metodologia, não são intuitivas, tampouco auxiliaram o processo.

Para suprir as nossas necessidades, fizemos uso de diagramas de i* para a coleta de objetivos e de diagramas de UML [48] para registrar informações de maneira mais intuitiva, a saber, no registro do protocolo de interações, usando seus diagramas de seqüência.

3 Tecnologias utilizadas

Neste capítulo iremos descrever algumas das tecnologias utilizadas durante o desenvolvimento do nosso sistema multiagente.

3.1 O framework Jade

Jade, ou Java Agent Development framework (framework para desenvolvimento de agentes em Java) é um framework de desenvolvimento de agentes com o objetivo de desenvolver SMA e aplicações que estejam de acordo com os padrões da FIPA para desenvolvimento de agentes inteligentes. Jade é composto de dois produtos principais: uma plataforma para agentes de acordo com os padrões da FIPA e um pacote para desenvolvimento de agentes em Java.

O framework Jade, possui uma série de ferramentas para facilitar o desenvolvimento de aplicações e a administração da plataforma, sendo elas: um agente de gerenciamento remoto, *RMA* (remote management agent) – uma plataforma gráfica para gerenciamento e controle da plataforma; o agente *dummy*, que pode ser utilizado para a composição de mensagens ACL e enviá-las e recebê-las entre outros agentes; o agente *sniffer*, que permite a captura do tráfego gerado entre os agentes; o *introspector*, que permite a monitoração do ciclo de um agente, incluindo suas mensagens trocadas e os comportamentos em execução; uma interface gráfica para o serviço de diretórios, *DF* (Directory Facilitator), permitindo a visualização e a alteração da base de conhecimento do serviço de páginas amarelas em tempo de execução; um *Agente de Gerenciamento de Logs*, pemritindo a alteração de configurações de log em tempo de execução, e um agente *Proxy*, que permite a execução de um gateway entre uma plataforma Jade e uma contexão TPC/IP comum.

A plataforma de agentes implementada por Jade, segue o padrão de referência da FIPA, contendo os elementos básicos, que são: um sistema de gerenciamento de agentes, um serviço de diretório e um sistema de transporte de mensagens, conforme

ilustrado na figura 3.1.

Esta plataforma não precisa estar necessariamente contida em uma única máquina, podendo estar distribuída em máquinas distintas, com sistemas operacionais e configurações distintas, sendo necessário apenas que possam executar uma máquina virtual Java. Jade permite que exista apenas um Gerenciador de Agentes (AMS – Agent Management System) gerenciando a plataforma, e, no caso de uma plataforma que se estenda por várias máquinas, o próprio framework se encarrega de manter a comunicação entre os containers e a utilização do AMS, conforme figura 3.2.

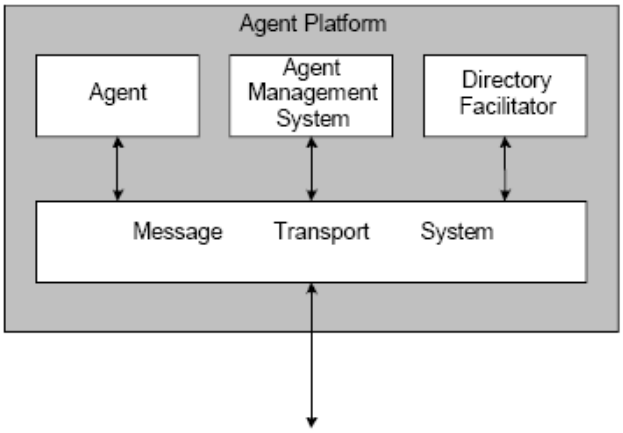

Figura 3.1 – *Arquitetura de referência para uma plataforma de agentes FIPA – adaptado de [27]*

Do ponto de vista do programador, a implementação de um agente no framework Jade é feito simplesmente se estendendo a classe *agent* já existente. Através do mecanismo de herança, os agentes implementados passam a ter as características necessárias para poder realizar as interações básicas com a plataforma de agentes (que inclui o registro, configuração, gerenciamento remoto, entre outros). Além disso, os agentes herdam também um conjunto de métodos básicos que podem ser utilizados para implementar a comportamento do agente (por exemplo, o envio e recebimento de mensagens, protocolos de interação padrões, o registro em vários domínios, etc.).

Os elementos básicos de um agente implementado através do framework Jade são: os comportamentos (behaviors), que implementam os comportamentos executados pelo agente; e a fila de mensagens, que gerencia o envio e o recebimento das mensagens individualmente de cada agente. A execução de um agente é controlada por

um *scheduler*, ou escalonador, que controla o agendamento da execução dos *behaviors* dos agentes.

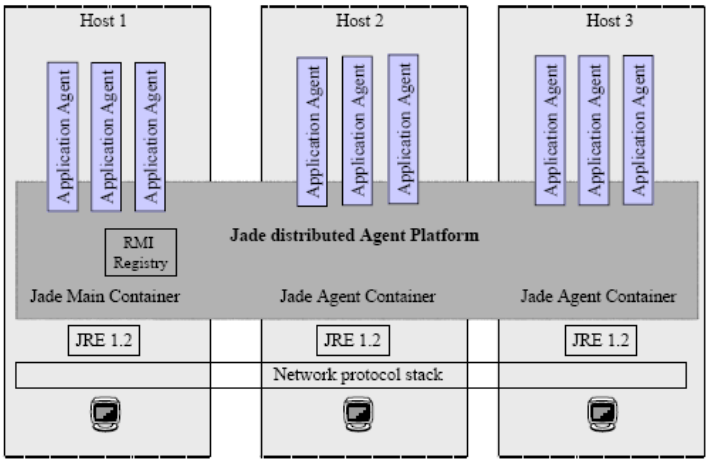

Figura 3.2 – *Plataforma de Agentes JADE, distribuída entre vários contêiners – adaptado de [26]*

4 O DBSitter Plus

Nosso trabalho foi composto dos seguintes passos:

1. Coleta de requisitos do problema do DBSitter;
2. Aplicação da metodologia escolhida, Gaia;
3. Desenvolvimento de um protótipo.

Para a coleta de requisitos do problema do DBSitter, utilizamos a metodologia i*, validando os resultados obtidos com profissionais da área de Banco de Dados. Uma vez coletados estes requisitos, passamos à aplicação da metodologia Gaia, e a geração dos artefatos relacionados por ela. Por último, implementamos um protótipo através da utilização dos artefatos gerados.

4.1 Requisitos

A metodologia de desenvolvimento de SMA Gaia, não prevê nenhum mecanismo para coleta de requisitos, pressupondo que os mesmo já estão definidos antes da sua aplicação. Em [17], é sugerida a utilização de métodos de coleta de requisitos utilizados atualmente, como casos de uso, ou a metodologia i*, visto que eles tem como objetivo, não a identificação de objetos do sistema, mas dos objetivos do mesmo.

A escolha dos diagramas i* [42] se deu pelas características do mesmo se adequar à nossa necessidade. Os diagramas i* possuem como característica a coleta de objetivos através de uma forma gráfica.

O processo utilizado para a geração dos diagramas i* podem ser observados no apêndice A, onde é descrito a identificação dos atores envolvidos no sistema e o caminho seguido até a definição do xaADB. Como nosso trabalho tem como foco a aplicação da metodologia Gaia, nós apresentamos o Diagrama de Raciocínio estratégico, figura 4.1, onde são apresentados os objetivos do DBSitter.

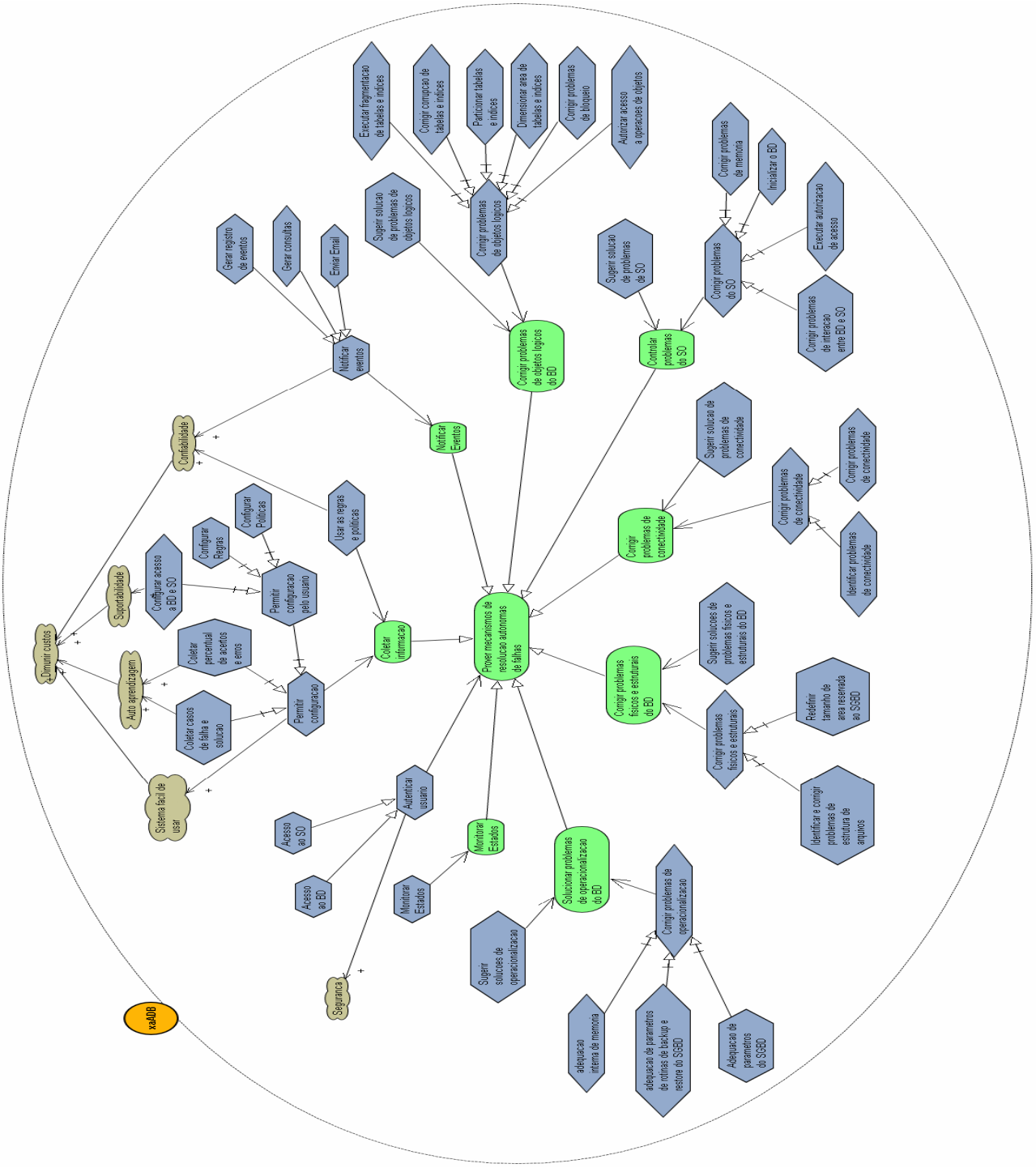

Figura 4.1 – *Diagrama de objetivos do xaADB*

Podemos ver no centro da figura 4.1 o objetivo principal do sistema que é prover mecanismos de resolução autônoma de falhas, e a sua decomposição em subobjetivos e tarefas. Os subobjetivos necessários para a provisão de mecanismos de resolução autônoma de falhas são: a coleta de informações; o monitoramento de estados, a solução de problemas de operacionalização do SGBD; a correção de problemas físicos e estruturais do SGBD, a correção de problemas de conectividade; o controle de problemas do Sistema Operacional (SO), a correção de problemas de objetos lógicos do SGBD e a notificação de eventos, como pode ser observado no diagrama. Por sua vez, cada subobjetivo foi decomposto nas tarefas necessárias para atingir este subobjetivos, caracterizados pelos hexágonos a eles ligados. Segue abaixo uma descrição dos subobjetivos:

- Correção de problemas de objetos lógico do SGBD pode ser alcançado através: da sugestão da solução para problemas de objetos lógicos, assim como da correção de problemas de objetos lógicos. Este último inclui a execução de desfragmentação de tabelas e índices, a correção da corrupção de tabelas e índices, o particionamento de tabelas e índices, o dimensionamento de área de tabela e índices, a correção de problemas de bloqueio e a autorização de acesso a operações de objetos.

- Controle de problemas de SO, pode ser atingido através da execução das tarefas: sugestão de solução de problemas de SO e da correção de problemas de SO. Este último pode ser decomposto nas seguintes tarefas: correção de problemas de interação entre SGBD e SO; execução de autorização de acesso; inicialização do SGBD e correção de problemas de memória.

- Correção de problemas de conectividade pode ser atingido através da execução das seguintes tarefas: sugestão para solução de problemas de conectividade e correção de problemas de conectividade. Este último é composto das tarefas: identificação de problemas de conectividade e correção de problemas de conectividade.

- Correção de problemas físicos e estruturais do SGBD pode ser alcançado através

das tarefas: sugestão de solução de problemas físicos e estruturais do SGBD e da correção de problemas físicos e estruturais. Este último pode ser decomposto nas tarefas: identificação e correção de problemas de estrutura de arquivos e da redefinição de tamanho de área reservada ao SGBD.

- Solução de problemas de operacionalização do SGBD pode ser obtido através das tarefas: sugestão de soluções de operacionalização e da correção de problemas de operacionalização. Este último pode ser decomposto nas tarefas: adequação interna de memória, adequação de parâmetros de rotinas de backup e restore do SGBD e na tarefa adequação de parâmetros do SGBD.

- Monitorar estados é realizado através da tarefa de monitoração de estados, enquanto o objetivo notificar eventos pode ser realizado através da tarefa notificar eventos, que pode ser decomposta em geração de registro de eventos, geração de consultas e envio de e-mail.

- Coletar informação pode ser realizado através das tarefas de uso das regras e políticas e da permissão de configuração. A permissão de configuração pode ser dividida nas subtarefas permissão de configuração pelo usuário, coleta de percentual de acertos e erros e coleta de casos de falha e solução. A permissão de configuração pelo usuário pode ser subdivida novamente nas tarefas configuração de políticas, configuração de regras e configuração de acesso a SGBD e SO.

4.2 A modelagem SMA

Uma vez definido os requisitos, passamos à modelagem através da metodologia Gaia. Como descrito pela metodologia, iniciamos pelo processo de análise, identificando potenciais suborganizações, descrevemos o ambiente através dos recursos que serão utilizados pelo nosso sistema, criamos um modelo de papéis preliminar, baseado nas habilidades necessárias, identificamos as interações entre os papéis através do modelo de interações preliminar e por último identificamos regras gerais de relacionamentos entre

papéis, entre interações e entre interações e papéis que seriam melhor capturadas através de regras organizacionais.

Durante o processo de definição de suborganizações, identificamos que algumas porções do sistema exibiam um comportamento especificamente orientado para a realização dos mesmos subobjetivos, agrupando-os nas seguintes suborganizações:

- Suborganização de Detecção e Correção de Problemas
- Suborganização de Notificações e Registros
- Suborganização de Aprendizagem e Sugestão

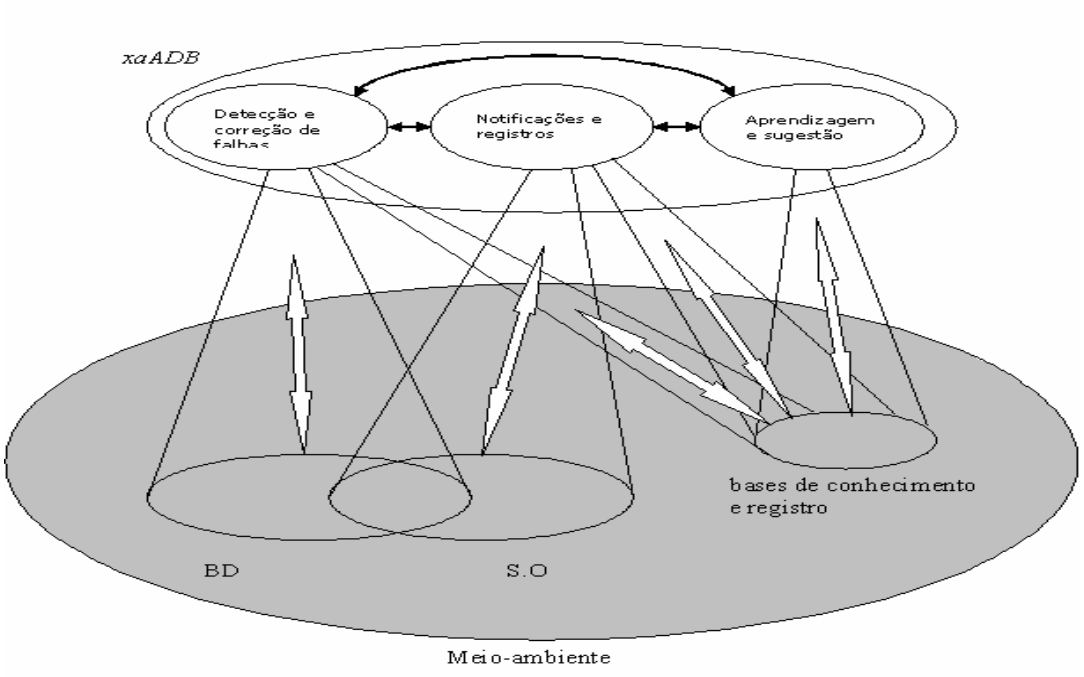

Figura 4.1 – *Diagrama de suborganizações do xaADB*

Após isto, passamos a descrever o ambiente. Gaia sugere tratar o ambiente em termos computacionais, descrevendo os recursos aos quais o nosso sistema precisa ter acesso para poder atingir os seus objetivos. Em sua forma mais simples pode ser visto como uma lista de recursos, associados com um nome simbólico e caracterizados pelos tipos de ações que os agentes podem realizar sobre eles. Para organizar melhor este processo, optamos por agregar os recursos de acordo com as suborganizações que foram identificadas.

Recursos da suborganização de Detecção e Correção de problemas:

- Informações sobre políticas organizacionais;
  - São informações contidas na base de conhecimento que regem o comportamento dos agente.
- Informações sobre regras de negócio;
  - São informações contidas na base de conhecimento, indicando, por exemplo, que determinados agentes apenas podem entrar em execução fora do horário comercial.
- Informações sobre o sistema operacional;
  - São as informações extraídas do SO, e.g.: processos em execução.;
- Estados do banco de dados;
  - São informações de estado do banco extraídas através de consultas aos dicionários de dados (catálogos do SGBD);
- Registros de log do SGBD;
  - São os arquivos de registros de ações e erros do SGBD monitorado.
- Base de conhecimento com ações de correção de falhas e estatísticas de ações realizadas;
  - É a base de conhecimento onde vão estar registrados os sintomas e as ações de correção para cada caso de falha.

Recursos da suborganização Notificações e Registros:

- Informações sobre políticas organizacionais;
  - São informações contidas na base de conhecimento que regem o comportamento dos agente. E.g.: Apenas o agente notificador pode enviar alertas para o usuário final.
- Log de ações;
  - O local onde vão estar armazenadas as informações das ações realizadas pelo DBSitter Plus.

Recursos da suborganização Aprendizagem e Sugestão:

- Base de conhecimento sobre políticas organizacionais;
  - São informações contidas na base de conhecimento que regem o comportamento dos agente. E.g.: Apenas o agente aprendiz é autorizado a manipulação de escrita na base de conhecimento.
- Base de conhecimento sobre regras de negócio;
  - São informações contidas na base de conhecimento, indicando, por exemplo, que determinados agentes apenas podem entrar em execução fora do horário comercial.
- Base de conhecimento com ações de correção de falhas e estatísticas de ações realizadas.
  - É a base de conhecimento onde vão estar registrados os sintomas e as ações de correção para cada caso de falha.

Para a identificação dos papéis do nosso sistema, detectamos todas as habilidades necessárias para a execução dos nossos objetivos, e transformamos essas habilidades em papéis. Gaia define papéis como sendo compostos de quatro atributos básicos: responsabilidades, permissões, atividades e protocolos.

Responsabilidades são os atributos chave, relacionados a um papel e determinam sua funcionalidade. Responsabilidades são classificadas em dois tipos: *Liveness* e *Safety*. *Liveness* diz o que o papel agrega de útil ou positivo ao sistema. Deve descrever atividades que um agente que implemente o papel deve desempenhar sob determinadas condições ambientais. *Safety* diz o que o papel deve prevenir ou desautorizar para que nada de ruim ou indesejado aconteça ao sistema. Assegura que um estado aceitável em ocorrências é mantido durante o ciclo de execução.

Para que as Responsabilidades sejam atendidas, um papel deve estabelecer um conjunto de permissões. As permissões representam o que o papel está permitido a fazer e quais recursos pode acessar. E atividades são tarefas que um papel realiza sem ter que interagir com outro agente. Protocolos são padrões específicos de interação, por exemplo, determinado tipo de leilão.

Com isso, identificamos os seguintes papéis preliminares em nosso sistema:

- Detector de erros de estruturas de controle de SGBD
  - Responsável pela monitoração das estruturas de controle do SGBD;
- Detector de erros de estruturas físicas de SGBD;
  - Responsável pela monitoração das estruturas físicas do SGBD;
- Detector de erros de estruturas lógicas de SGBD;
  - Responsável pela monitoração das estruturas lógicas do SGBD;
- Corretor de falhas de estruturas de controle de SGBD;
  - Responsável pela atuação nas estruturas de controle do SGBD;
- Corretor de falhas de estruturas físicas de SGBD;
  - Responsável pela atuação nas estruturas físicas do SGBD;
- Corretor de falhas de estruturas lógicas de SGBD;
  - Responsável pela atuação nas estruturas lógicas do SGBD;
- Detector de erros de S.O.;
  - Responsável pela monitoração do estado do SO;
- Corretor de falhas de S.O.;
  - Responsável pela atuação no estado do SO;;
- Detector de erros de Rede;
  - Responsável pela monitoração do estado da rede;
- Corretor de falhas de Rede;
  - Responsável pela atuação no estado da rede;
- Notificador
  - Responsável pela notificação de erros ao ABD;
- Aprendiz
  - Responsável pelo aprendizado do sistema;

Na tabela 4.1, podemos verificar o modelo sugerido por Gaia para armazenamento das informações contidas nos papéis.

Papel: MonitorControlDB	
Descrição	Esse papel preliminar irá monitorar constantemente os estados das estruturas e controle do banco de dados alvo, monitorando processos do SGBD no S.O., em arquivos de logs (traces) de SGBD ou realizando consultas em dicionários de dados. Caso solicitado, ele irá se comunicar com outros papéis que requisitem informações, passando dados sobre o estado atual da estruturas monitoradas do banco.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, MonitoringDB, registerServiceDirectory
Permissões	Reads DBState *// O estado do banco de dados* Reads OSState *// O estado dos processo do SO*
Responsabilidades	
Liveness	MONITORCONTROLDB = registerServiceDirectory.((ReceiveInfRequest . ProccessInfRequest . AnswerInfRequest) | (MonitoringDB)) $^{\omega}$
Safety	Ter acesso completo ao servidor, incluindo aos processos que estão sendo executados pelo SO, processos do SGBD e ao SGBD alvo.

Tabela 4.1 - *Detector de erros de estruturas de controle de SGBD*

No apêndice B, podemos encontrar o modelo de papéis com a descrição completa de todos os papéis identificados na fase de análise.

Após identificados os papéis preliminares, passamos a identificar o modelo de interações preliminares, que descreve os relacionamentos entre os papéis. Nós identificamos a seguinte lista preliminar de interações:

- ReceiveInfRequest,
  - A comunicação utilizada para extrair informações dos detectores.
- AnswerInfRequest,
  - A comunicação utilizada para enviar informações dos detectores.
- AskForInformation,
  - A comunicação utilizada para requisitar informações de status dos detectores.
- ReceiveInformation,
  - A comunicação utilizada para o recebimento de informações de status.
- ReceiveRequest,
  - A comunicação utilizada para requisitar uma atuação.
- AnswerRequest,
  - A comunicação utilizada para retornar o status de uma requisição de atuação.
- ReceiveEpisodeFeedback,
  - A comunicação utilizada para receber feedback dos episódios de correção.

Nome do Protocolo: **ReceiveInfRequest**		
Iniciador: Todos atuadores	Parceiro: Todos detectores	Entrada:**Requisição para informação.**
Descrição: Mensagens utilizada para extrair informações dos detectores.		Saída: **Erro ou msg com status do objeto monitorado.**

Tabela 4.1 - Exemplo de definição preliminar de protocolo

Na tabela 4.1, podemos verificar o modelo sugerido por Gaia para o armazenamento das informaçções do protocolo de interações. A lista completa dos protocolos de interações preliminares pode ser vista no apêndice B.

Uma vez identificados os agentes e os protocolos, passamos a capturar as regras organizacionais, que são relacionamentos entre papéis, entre protocolos e entre papéis e protocolos. As regras organizacionais são como as responsabilidades individuais de cada agente, entretanto, aplicadas à organização como um todo. Ou seja, podemos ter regras de liveness, que indicam como a organização deve se comportar com o passar do tempo, e regras de safety, que são regras que capturam as invariantes do sistema.

PassarInformacoes(Monitor) → EnviarEmailDeNotficacao(Notificador)

Uma vez cumpridas estas tarefas, passamos à fase de projeto, onde iremos fazer uso das informações coletadas. Enquanto na fase de análise nosso foco estava na compreensão do que o SMA deverá fazer, na fase de projeto começamos a tomar decisões sobre as características finais do SMA.

De acordo com Gaia [17], para determinar a melhor escolha do projeto de estrutura organizacional, deve-se considerar a influência de fatores como: regime de controle (particionamento de trabalho, especialização de trabalho, modelos baseados em mercado, etc), complexidade computacional e de coordenação, regras organizacionais, influencia da organização no mundo real e simplicidade. A **topologia** (e.g.: *centralizada*, *coleção de plataformas* (*peers*), *hierárquica*, *hierárquica multinível*, *topologias compostas*

complexas, topologias em redes híbridas) é então selecionada baseado nos pesos para cada um desses fatores.

Essa escolha influenciará sobremaneira todas as fases subseqüentes do projeto, sendo portanto uma etapa crítica no processo. Esta análise deve ter em mente respeitar as regras organizacionais levantadas e minimizar a distância para a organização do mundo real.

Por se adequar melhor ao problema em questão, escolhemos a estrutura organizacional de topologia **hierárquica**, onde um líder assume as responsabilidades de coordenação. A figura 4.2 apresenta a organização escolhida. Essa decisão é corroborada com a intenção de utilização do método de coordenação de agentes PGP (*Partial Global Planning*), onde agentes setoriais possuem planos locais de resolução de atividades, mas devem respeitar as diretrizes de um plano global seguido por todos de maneira cooperativa e determinado por um agente coordenador.

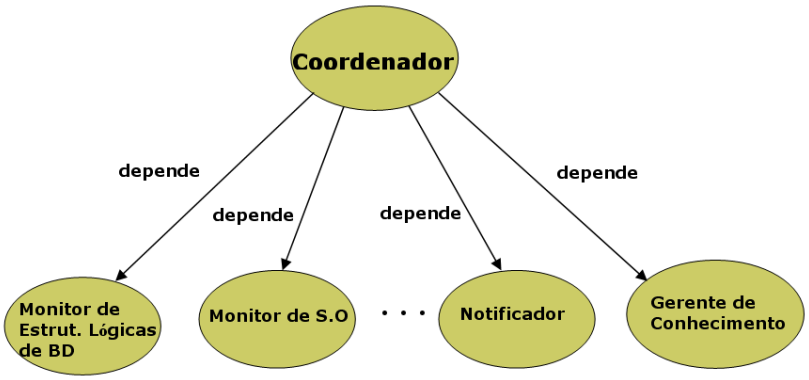

Figura 4.2 – *Topologia da estrutura organizacional hierárquica*

Após a escolha do modelo organizacional, passamos ao refino do modelo de papéis. Como evolução dos papéis preliminares foi criado o papel de Coordenador de Atividades. Os papéis de Detector e Corretor de erros setoriais, por possuírem percepções similares e atribuições complementares, foram fundidos em papéis Monitores, que englobam as suas responsabilidades. Apenas o papel de coordenador tem habilidade de solicitar alteração na base de conhecimento. O papel Aprendiz teve sua capacidade expandida para gerente de conhecimento.

Abaixo segue uma descrição sucinta dos papéis definitivos concebidos e uma descrição formal dos papéis segundo os padrões da metodologia Gaia encontra-se listada no Apêndice D.

- Monitor de estruturas de controle de BD
    - Junção dos papéis preliminares detector e corretor de erros de estruturas de controle de BD;
- Monitor de estruturas físicas de BD
    - Junção dos papéis preliminares detector e corretor de erros de estruturas físicas de BD;
- Monitor de estruturas lógicas de BD
    - Junção dos papéis preliminares detector e corretor de erros de estruturas lógicas de BD;
- Monitor de SO
    - Junção dos papéis preliminares detector e corretor de erros de S.O.;
- Monitor de rede
    - Junção dos papéis preliminares detector e corretor de erros de rede;
- Notificador
    - Não houve alteração em relação ao papel preliminar;
- Gerente de Conhecimento
    - Expansão das responsabilidades do papel preliminar, que passa a assumir uma função de *wrapper* de toda a base de conhecimento. É o único papel a interagir com a base de conhecimento para escrita. Atende aos outros papéis por meio de solicitação.
- Coordenador
    - Criado esse papel para realizar a coordenação de todo o processo de identificação, correção, notificação e registro de feedback de atuação para os diversos erros ocorridos nas suborganizações de detecção e correção de falhas (estruturas de controle de SGBD, lógicas de SGBD, físicas de SBBD, S.O. e Redes). Também se responsabiliza pela concepção de todos os planos para correção de episódios de erro e gerenciamento de tarefas que envolvam a perticipação de mais de um agente.

Partindo do modelo de interações preliminar, e à luz do novo modelo de papéis pode-se formatar o modelo de interações definitivo.

Exibimos abaixo, na Figura 4.3, um diagrama de interação concebido nos moldes do diagrama de seqüência Unified Modelling Language (UML). Este diagrama exibe o modelo de interação do caso de falha implementado na prova de conceito desenvolvida.

O diagrama exibido descreve a atividade de correção de falha de dimensionamento de memória compartilhada no BD Oracle. Para solucionar a falha, o papel monitor de estruturas de controle deve passar seu plano parcial de resolução para o papel coordenador junto com a informação do acontecimento de episódio de falha. O papel coordenador em seguida deve consultar regras de negócio e a base de conhecimento (encapsulada em um papel) para montar o plano global de resolução.

Para resolver o episodio, o agente monitor deve seguir o plano global e seu plano parcial. Para tanto, deve solicitar ao papel monitor de BD para que retire o BD do ar. Esse papel deve confirmar que há um plano prevendo retirada de BD do ar nas condições correntes. Uma vez confirmado e o BD fora do ar, o agente monitor de estrutura de controle altera o arquivo de configuração do BD aumentando a memória disponível e em seguida solicita que o papel monitor de BD recoloque o BD no ar.

Uma checagem de efetividade deve ser feita e esse feedback repassado ao agente coordenador que por sua vez registrará o episódio e resultado na base de conhecimento.

Com isto, finalizamos a fase de projeto e passamos para a fase de projeto detalhado, onde iremos definir o modelo de agentes e o modelo de serviços.

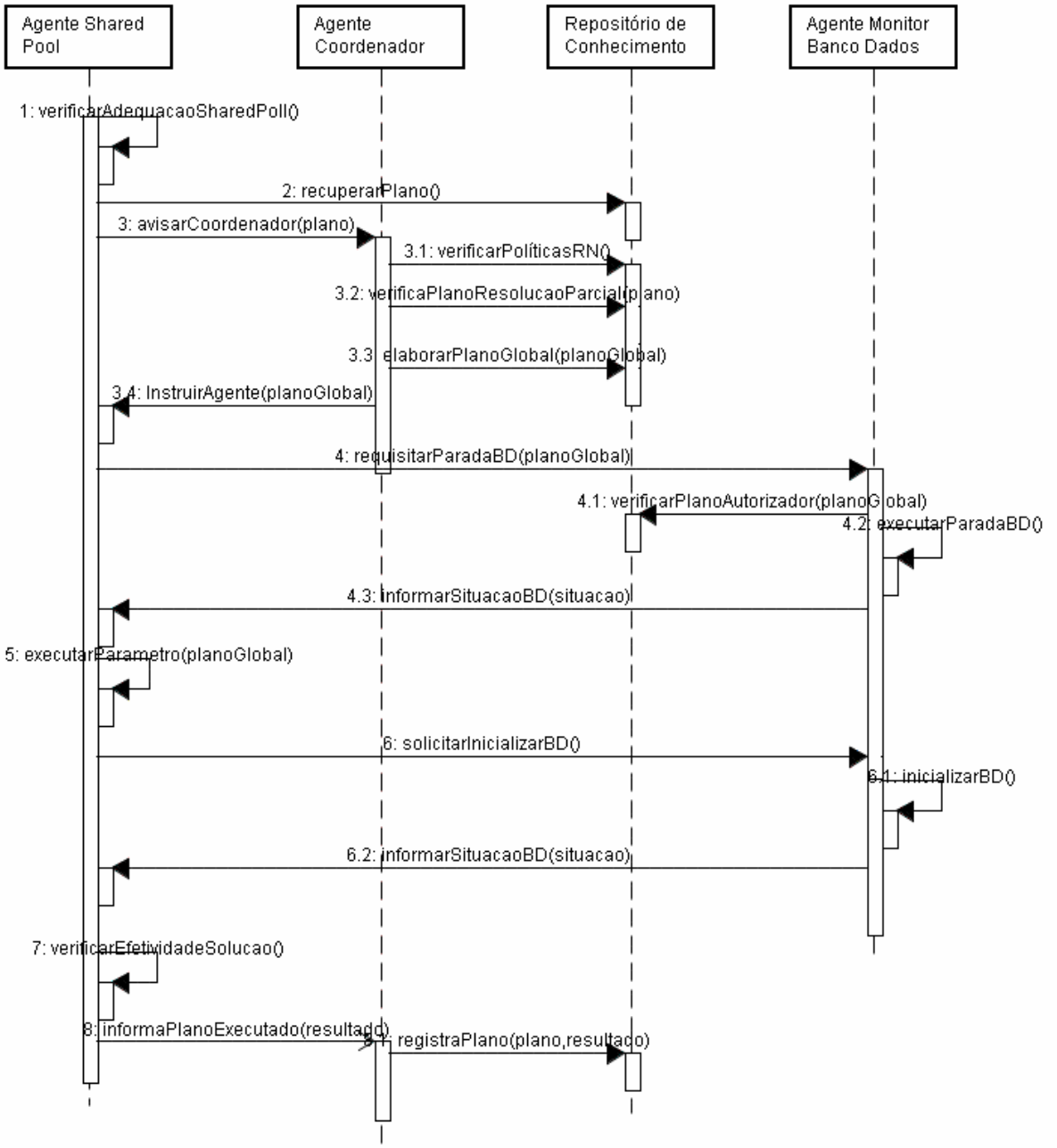

Figura 4.3 – Modelo de interação representado através de diagrama de seqüência UML

O modelo de agentes detalha que agentes irão desempenhar quais papéis, assim como a quantidade de instâncias deles. Já o modelo de serviços, como o próprio nome sugere, tem como objetivo identificar os serviços associados a cada agente.

Podemos observar na Figura 4.4, como os papéis serão desempenhados por agentes.

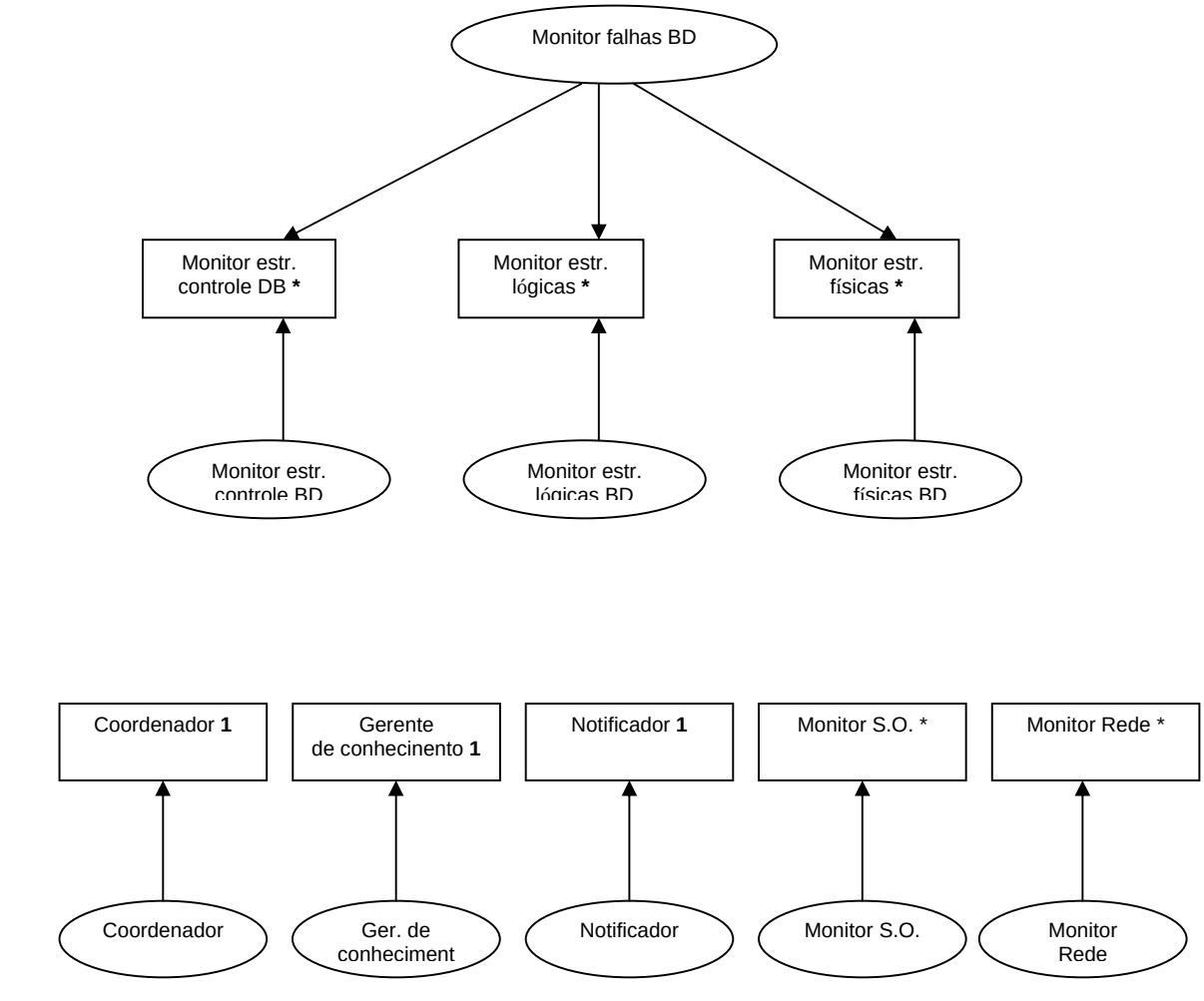

Figura 4.4 – *Modelo de Agentes*

Na Tabela 4.2, podemos verificar alguns dos serviços identificados. O apêndice E contêm a lista dos serviços implementados pelo nosso estudo de caso.

Serviço	**Resolução de episódio de falha**	**Registro de feedback**
Entradas	Identificação do plano global	Identificação do tipo de falha e medida de eficiência do episódio
Saídas	Status de realização concluída / Medida de efetividade	Status de realização concluída
Pré-condições	Autorização do coordenador	Feedback coletado
Pós-condições	Feedback coletado	-

Tabela 4.2 – *Exemplo de Modelo de serviços*

4.3 Experimentos e resultados

Desenvolvemos, como prova de conceito, uma implementação dos agentes *Agente Shared Pool*, *Agente Monitor de Banco de Dados*, *Repositório de Conhecimento* e *Agente Coordenador*.

Através da utilização da plataforma JADE, geramos classes que implementam cada um dos agentes identificados. Estes agentes tem seus comportamementos implementados através de *behaviors*. Durante o desenvolvimento fizemos uso dos artefatos gerados para orientar o desenvolvimento, em especial dos diagramas de seqüência e dos modelos de papéis definitivos. Estes nos auxiliaram a identificar os comportamentos dos agentes.

Através da ferramente *sniffer* disponível na plataforma JADE, monitoramos a comunicação, verificando que a mesma ocorreu da forma esperada. O comportamento dos agentes atuadores, *Agente Shared Pool e Agente Monitor de Banco de Dados*, também ocorreu corretamente, sendo monitorados através da ferramenta *introspector*.

O mecanismo de coordenação está sendo objeto de estudo de uma pesquisa em andamento, de tal modo que apenas implementamos uma simulação do seu funcionamento, visto que não é o objeto principal de nosso trabalho.

Consideramos ser o caso de uso implementado bastante representativo, visto que contempla comunicação entre agentes monitores entre si e com o coordenador, registro

de execuções e utilização da base de conhecimento.

Através desse experimento, pudemos comprovar o auxílio gerado por um processo de desenvolvimento de SMA, além de comprovar a viabilidade de implementação do nosso estudo do DBSitter.

4.4 Considerações finais

A nossa implementação do DBSitter Plus, propiciou a adequação aos padrões da FIPA, facilitando extensões do mesmo ou da sua comunicação com outros SMA que também implementem os padrões da FIPA.

Durante o desenvolvimento, verificamos a validade da utilização de uma metodologia de desenvolvimento SMA, inclusive devido ao fato que documentamos nossas decisões através dos artefatos da mesma, permitindo sua reutilização.

5 Conclusões e trabalhos futuros

5.1 Contribuições

Através deste trabalho, aumentamos a compatibilidade do DBSitter, através da sua adequação às padronizações da FIPA. Para tanto revisamos os conceitos originais através da modelagem do projeto utilizando a metodologia GAIA, específica para SMA.

Comprovamos a viabilidade da implementação das novas características, através do desenvolvimento, com o *framework* JADE, de um caso representativo de resolucao automática de falha em SGBD.

5.2 Dificuldades

Algumas das dificuldades encontradas durante o desenvolvimento deste trabalho se referem à utilização da notação da metodologia GAIA e da reutilização de código existente. Em relação à metodologia GAIA, percebemos a falta da especificação de uma fase de requisitos, e fizemos uso da modelagem i*. Além disso, sentimos a necessidade da utilização de diagramas mais expressivos para o processo, adotando diagramas existentes em UML. Em relação à reutilização de código, a nossa proposta para alteração da padronização dos agentes, gerou uma mudança no modelo de comunicação que inviabilizou o reaproveitamento de uma parcela significativa do código existente.

5.3 Trabalhos futuros

Como trabalhos futuros, poderíamos citar o desenvolvimento de protótipos para diferentes versões de SGBD, com por exemplo o PostgresSQL, mySQL ou SQLServer.

Desenvolvimento de novos casos de resolução de falha, com diferentes graus de complexidade, assim como a utilização de diferentes implementações para o mecanismo de coordenação.

Referências bibliográficas

[1] Carneiro A., "DBSitter: Um Sistema Inteligente para Gerenciamento de SGBD", 2005

[2] Carneiro A., Passos R., Belian R., Costa T., Tedesco P., Salgado A. C, "DBSitter: An Intelligent Tool for Database Administration", 2004

[3] Kephart, J.O., Chess, D. M., The vision of autonomic computing, disponível em: http://www.research.ibm.com/autonomic/research/papers/AC_Vision_Computer_Jan_2003.pdf, acesso em 01/10/2006.

[4] Horn, P., Autonomic Computing: IBM's Perspective on the State of Information Technology, disponível em: http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf, acesso em 01/10/2006.

[5] Ganek, A. G., Corbi T. A., "The Dawning of Autonomic Computing Era", 2003

[6] Johnston-Watt, Duncan, "Under New Management: Autonomic Computing is Revolutionizing The Way We Manage Complex Systems", 2006

[7] Lin, P., MacArthur, A., Leaney, J., Defining Autonomic Computing - A Software Engineering Perspective

[8] IBM, "An Architectural blueprint for Autonomic Computing", 2006

[9] Koehler, J., Giblin C., Gantenbein, D., Hauser R., "On Autonomic Computing Architectures", 2003

[10] Elnaffar, S., Powley W., Benoit D., Martin P., "Today's DBMSs: How autonomic are they?", 2003

[11] Lifschitz, S. and Macêdo, J. (2004). Agent-based Databases and Parallel Join Load

Balancing, chapter 6, pages 69–88. Sistemas de Informacíon e Ingeniería de Software: Temas Selectos - Editora Centro Estudios Informatica, Univ. Simon Bolivar (Merida) Venezuela.

[12] Lifschitz, Design and Implementation of a Global Self-tuning architecture

[13] Foundation for Intelligent Physical Agents – FIPA, disponível em: http://www.fipa.org, acesso em 01/10/2006.

[14] White, S.R. Hanson, J.E. Whalley, I. Chess, D.M. Kephart, J.O., "An architectural approach to autonomic computing", 2004

[15] Murilo Juchem, M; Bastos R. M., Engenharia de Sistemas Multiagentes: Uma Investigação sobre o Estado da Arte, 2001

[16] Wagner, G., "The Agent-Oriented-Relationship Metamodel:Towards a Unified View of The State and Behavior", 2003

[17] Zamborelli, F.; Jennigs N. R.; Wooldridge, M., Developing Multiagent Systems: The Gaia Methodology", 2003

[18] Wooldridge M., "An Introduction to MultiAgent Systems"

[19] Moraïtis P., Petraki E., Spanoudakis N., ”Engineering JADE Agents with the Gaia Methodology”, 2003

[20] Wooldridge M., Jennings, N. R.. Pitfalls of Agent-Oriented Development

[21] Jennings, N. R., Agent-based computing: promise and perils

[22] Lin, P., MacArthur, A., Leaney, J., Defining autonomic Computing: a software engineering perspective

[23] Foundation for Intelligent Physical Agents – FIPA, FIPA Communicative Act Librarry Specification - SC00037J

[24] Foundation for Intelligent Physical Agents – FIPA, FIPA ACL Message Structure Specification - SC00061G

[25] Foundation for Intelligent Physical Agents – FIPA, FIPA Abstract Architecture Specification - SC00001L

[26] JADE Administrator's Guide, disponível em: http://jade.cselt.it, acesso em 01/10/2006.

[27] JADE Programmer's guide, disponível em: http://jade.cselt.it, acesso em 01/10/2006.

[28] Nikraz, M., Caire, G. Bahri, P. A., A methodology for the analysis and design of multi-agent systems using JADE

[29] Jennings, N. R., Wooldridge, M., Applications of intelligent agents

[30] Moraitis, P., Spanoudakis, N. I., Combining Gaia and JADE for multi-agent systems development

[31] Bauer, B., Müller, J. P., Odell, J., Agent UML: A formalism for specifying multiagent interaction

[32] Odell, J., Parunak, H. V. D., Bauer, B., Extending UML for agents

[33] Silva, V. T., Lucena, C. J. P., MAS-ML: A multi-agent system modeling language

[34] Wagner, G., The agent-object-relationship metamodel: towards a unified view of state and behavior

[35] Wagner, G., Taveter, K., Towards radical agent-oriented software engineering processes based on AOR modeling

[36] DeLoach, S. A., Analysis and design using MaSE and agentTool

[37] Padgham, L. Winikoff, M., Prometheus: A pragmatic methodology for engineering intelligent agents

[38] Java Agent DEvelopment Framework, disponível em: http://jade.cselt.it, acesso em 01/10/2006.

[39] Franklin, S. and Graesser, A., Is it an Agent, or just a Program? : A Taxonomy for Autonomous Agents, 1995.

[40] Weiss, G., Multiagent Systems: A modern approach to distributed artificial intelligence – The MIT Press, 1999.

[41] Jennings, N. R., On agent-based software engineering.

[42] Yu, E. S. K., Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering

[43] Russell , Norvig, Artificial Intelligence, a Modern Approach

[44] Moore, Gordon E.: Cramming more components onto integrated circuits – Electronics, Volume 38, Number 8, April 19, 1965

[45] J. P. Bigus, J. L. Hellerstein, T. S. Jayram, e M. S. Squillante. Auto tune: A generic agent for automated performance tuning. In Proceedings of the International Conference and Exhibition on the Pratical Application of Intelligente Agents and Multi-Agent Systems (PAAM), páginas 33 - 52. 2000.

[46] Y Diao, J. L. Hellerstein, S. Parekh, e J. P. Bigus. Managing web server performance with autotune agents. IBM Systems Journal, 42 (1): 136 - 149, 2003.

[47] J. A. F. Macêdo. Agent-based DMBSs study. Tese de mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), 2000

[48] R. L. C. Costa e S. Lifschitz. Index self-tuning and agent-based databases. In Proceedings of the Latin-American Conference on Informatics (CLEI), páginas 76, Abastracts Proceedings. 2002.

[49] UML Resource Page, disponível em: http://www.uml.org/, acesso em 01/10/2006.

[50] Silva, M. J., Sarmento, L. C. M., Modelagem de um Sistema Multiagentes utilizando TROPOS, disponível em: http://www.cin.ufpe.br/~in1096/2006-1/, acesso em 01/10/2006.

[51] Maciel, P. R. M., Robertson Novelino Ferraz, R. N., Utilizando a Metodologia GAIA para Modelagem do Sistema Arquitetural xaADB, disponível em: http://www.cin.ufpe.br/~in1096/2006-1/, acesso em 01/10/2006.

Apêndice A – Especificação de Requisitos e Diagrama i*

Seguem abaixo os diagramas gerados durante a especificação de requisitos do xaADB. Na figura A.1, podemos observar o diagrama de atores. Os atores identificados são cliente e dba, e este diagrama tem identificados suas interdependências e seus objetivos pessoais. O cliente tem o objetivo de Aperfeiçoar o Gerenciamento de Dados e o objetivo "soft" de Diminuir Custos. O cliente depende do dba para alcançar o objetivo de Administrar Dados e os objetivos "soft" Eficiencia na Administração de Dados e Manter Bom Nível Continuidade Serviços. O dba depende do cliente para melhorar a qualidade do trabalho.

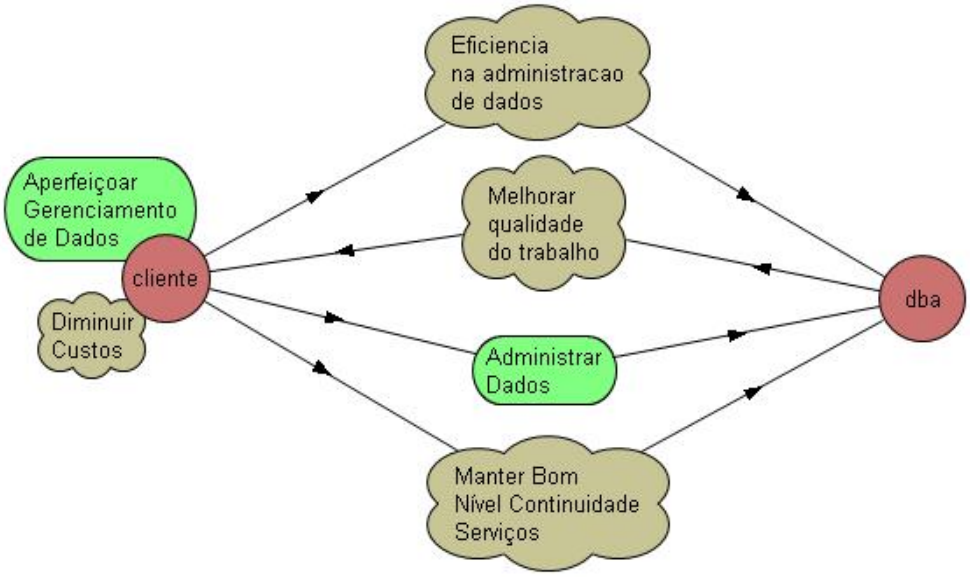

Figura A.1 – *Diagrama de atores*

A figura A.2 mostra a análise do raciocínio do cliente. Durante o processo análise de requisitos, concluiu-se que o objetivo "Aperfeiçoar Gerenciamento de Dados" pode ser alcançado através do plano "Avaliar Soluções" e que este plano contribui positivamente para "Melhorar a Qualidade do Trabalho".

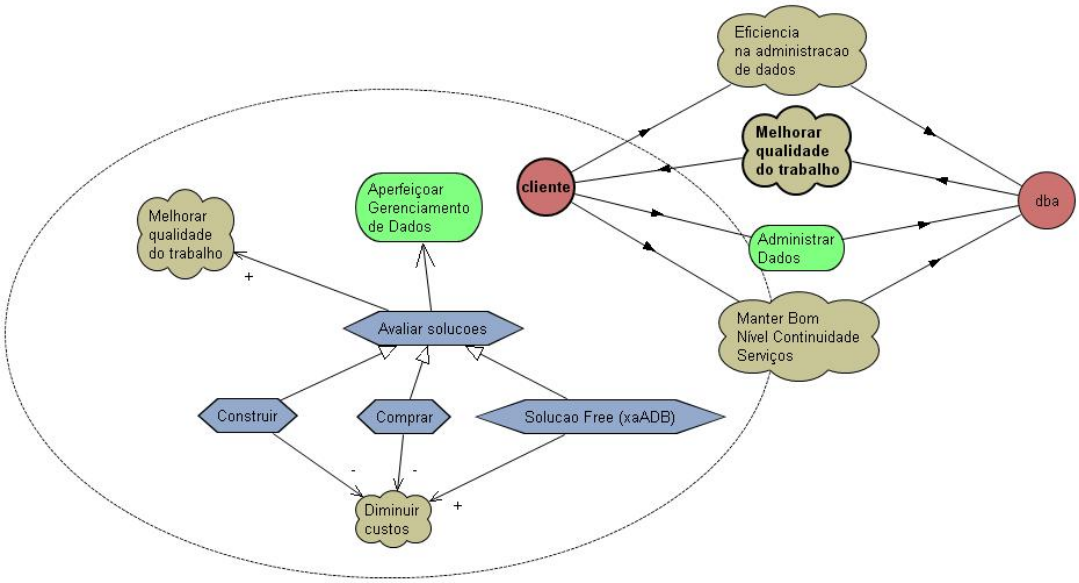

Figura A.2 – *Diagrama de objetivos do ator Cliente*

A figura A.3 mostra a análise do raciocínio do dba. Concluiu-se que o objetivo "Administrar os Dados" pode ser alcançado tanto com o objetivo "Automatizar Tarefas com o uso de SMA" quanto com o objetivo "Administração Convencional". Também se considerou que "Automatizar as Tarefas com o uso de SMA" contribui positivamente para "Manter Bom Nível Continuidade Serviços". Um meio para operacionalizar o objetivo "Automatizar as Tarefas com o uso de SMA" é "Usar sistema xaADB". "Automatizar as Tarefas com o uso de SMA" contribui mais positivamente na "Eficiência na Administração de Dados" que a "Administração Convencional".

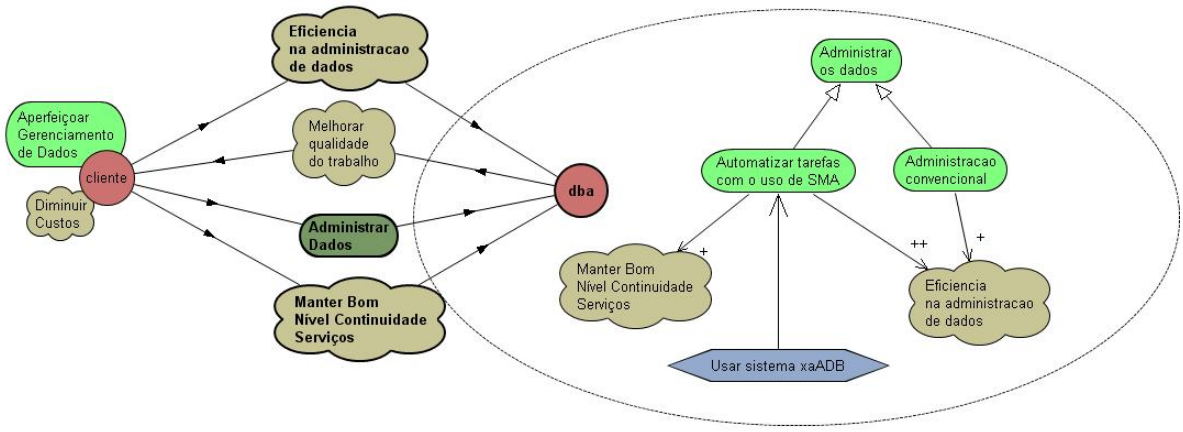

Figura A.3 – *Diagrama de objetivos do ator dba*

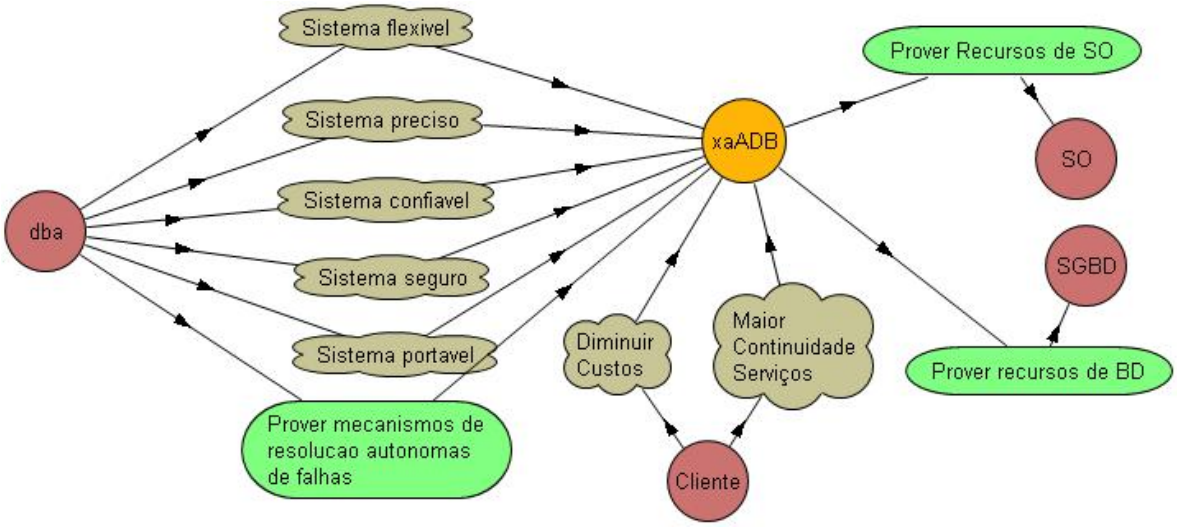

Figura A.4 – *Diagrama de atores estendido*

Na figura A.4 podemos ver o Diagrama de atores (stakeholders) ou Diagrama de Dependência estratégica (SD) estendido com o ator xaADB representando o sistema que será modelado. Neste diagrama, podemos perceber a inclusão do ator xaADB representando o sistema que será modelado e identificou as dependências dos outros atores da organização em relação a ele. Podemos ver na figura A.4 que o ator dba depende do ator xaADB para alcançar o objetivo de “Prover mecanismos de resolução autônoma de falhas” e que este atende aos objetivos soft de “Flexibilidade”, “Precisão”, “Confiabilidade”, “Segurança” e “Portabilidade”. Já o ator Cliente depende do ator xaADB para alcançar os objetivos soft de “Diminuir Custos” e “Maior Continuidade Serviços”.

Apêndice B – Papéis preliminares

Apresentamos a seguir os papéis preliminares identificados durante a fase de análise de Gaia.

Detector de erros de estruturas de controle de SGBD

Papel: MonitorControlDB	
Descrição	Esse papel preliminar irá monitorar constantemente os estados das estruturas e controle do banco de dados alvo, monitorando processos do SGBD no S.O., em arquivos de logs (traces) de SGBD ou realizando consultas em dicionários de dados. Caso solicitado, ele irá se comunicar com outros papéis que requisitem informações, passando dados sobre o estado atual da estruturas monitoradas do banco.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, MonitoringDB, registerServiceDirectory
Permissões	Reads DBState *// O estado do banco de dados* Reads OSState *// O estado dos processo do SO*
Responsabilidades	
Liveness	MONITORCONTROLDB = registerServiceDirectory.((ReceiveInfRequest . ProccessInfRequest . AnswerInfRequest) | (MonitoringDB)) $^{\omega}$
Safety	Ter acesso completo ao servidor, incluindo aos processos que estão sendo executados pelo SO, processos do SGBD e ao SGBD alvo.

Detector de erros de estruturas físicas de SGBD

Papel: MonitorPhysicalDB	
Descrição	Esse papel preliminar irá monitorar constantemente os estados das estruturas físicas do banco de dados alvo em arquivos de logs (traces) de SGBD, realizando consultas em dicionários de dados e verificando estados de estruturas de memória em disco (via S.O.). Caso solicitado, ele irá se comunicar com outros papéis que requisitem informações, passando dados sobre o estado atual do banco.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, MonitoringDB, registerServiceDirectory
Permissões	Reads DBState *// O estado do banco de dados* Reads OSState *// O estado dos processo e áreas do SO*
Responsabilidades	
Liveness	MONITORPHYSICALDB = registerServiceDirectory.((ReceiveInfRequest . ProccessInfRequest . AnswerInfRequest) | (MonitoringDB)) $^{\omega}$
Safety	Ter acesso completo ao servidor, incluindo aos processos que estão sendo executados pelo SO, áreas de memória em disco e ao SGBD alvo.

Detector de erros de estruturas lógicas de SGBD

Papel: MonitorLogicalDB	
Descrição	Esse papel preliminar irá monitorar constantemente os estados das estruturas lógicas do banco de dados alvo em arquivos de logs (traces) de SGBD ou realizando consultas em dicionários de dados. Caso solicitado, ele irá se comunicar com outros papéis que requisitem informações, passando dados sobre o estado atual do banco.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, ReceiveInformation, MonitoringDB, registerServiceDirectory
Permissões	Reads DBState *// O estado do banco de dados* Reads OSState *// O estado dos processos e áreas do SO*
Responsabilidades	
Liveness	MONITORLOGICALDB = registerServiceDirectory.((ReceiveInfRequest . ProccessInfRequest . AnswerInfRequest) | (MonitoringDB) | (ReceiveInformation)) $^{\omega}$
Safety	Ter acesso completo ao servidor, incluindo aos processos que estão sendo executados pelo SO, áreas de memória e dicionário de dados do SGBD alvo.

Corretor de falhas de estruturas de controle de SGBD

Papel: HealingDB	
Descrição	Esse papel irá executar ações que visam recuperar o estado do banco de dados e ações de correção de falhas. Eventualmente, também poderá requisitar informações de outros agentes.
Protocolos e Atividades	ReceiveRequest, ProccessRequest, AnswerRequest, AskForInformation, ReceiveInformation, SendEpisodeFeedback, HealingDB, registerServiceDirectory
Permissões	Reads KnowledgeBase // lê estratégias de correção de problemas Changes DBState *// Realiza operações no SGBD*
Responsabilidades	
Liveness	HEALINGDB = registerServiceDirectory ((ReceiveRequest.ProccessRequest.AnswerRequest) | (HealingDB) | (AskForInformation) | (ReceiveInformation) | (SendEpisodeFeedback)) $^{\omega}$
Safety	Ter acesso completo ao servidor, incluindo aos processos que estão sendo executados pelo SO e ao SGBD alvo.

Corretor de falhas de estruturas físicas de SGBD

Papel: HealingDB	
Descrição	Esse papel irá executar ações que visam recuperar o estado do banco de dados e ações de correção de falhas. Eventualmente, também poderá requisitar informações de outros agentes.
Protocolos e Atividades	ReceiveRequest, ProccessRequest, AnswerRequest, AskForInformation, ReceiveInformation, SendEpisodeFeedback, HealingDB, registerServiceDirectory
Permissões	Reads KnowledgeBase // lê estratégias de correção de problemas Changes DBState *// Realiza operações no SGBD*
Responsabilidades	
Liveness	HEALINGDB = registerServiceDirectory ((ReceiveRequest.ProccessRequest.AnswerRequest) | (HealingDB) | (AskForInformation) | (ReceiveInformation) | (SendEpisodeFeedback)) $^{\omega}$
Safety	Ter acesso completo ao servidor, incluindo aos processos que estão sendo executados pelo SO e ao SGBD alvo.

Corretor de falhas de estruturas lógicas de SGBD

Papel: HealingDB	
Descrição	Esse papel irá executar ações que visam recuperar o estado do banco de dados e ações de correção de falhas. Eventualmente, também poderá requisitar informações de outros agentes.
Protocolos e Atividades	ReceiveRequest, ProccessRequest, AnswerRequest, AskForInformation, ReceiveInformation, SendEpisodeFeedback, HealingDB, registerServiceDirectory
Permissões	Reads KnowledgeBase // lê estratégias de correção de problemas Changes DBState *// Realiza operações no SGBD*
Responsabilidades	
Liveness	HEALINGDB = registerServiceDirectory ((ReceiveRequest.ProccessRequest.AnswerRequest) | (HealingDB) | (AskForInformation) | (ReceiveInformation) | (SendEpisodeFeedback)) $^{\omega}$
Safety	Ter acesso completo ao servidor, incluindo aos processos que estão sendo executados pelo SO e ao SGBD alvo.

Detector de erros de SO

Papel: MonitorSO	
Descrição	Esse papel preliminar irá monitorar os estados do Sistema Operacional, através da aquisição de informações em variáveis de ambiente, processos e áreas de armazenamento. Irá se comunicar com papéis que requisitem informações sobre os estados do monitorados.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, MonitoringOS, registerServiceDirectory
Permissões	Reads OSState *// estado do S.O.* Changes OSState *// operações de alteração de elementos do S.O.*
Responsabilidades	
Liveness	MONITOROS = registerServiceDirectory ((ReceiveRequest . ProccessRequest . AnswerRequest) | (MonitoringSO))$^{\omega}$
Safety	Ter acesso completo aos recursos do sistema operacional

Corretor de falhas de SO

Papel: HealingSO	
Descrição	Esse papel irá executar ações que visam recuperar o estado de elementos do S.O.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, AskForInformation, ReceiveInformation, SendEpisodeFeedback, HealingOS, registerServiceDirectory
Permissões	Reads KnowledgeBase // lê estratégias de correção de problemas Changes SO State *// Realiza operações no S.O.*
Responsabilidades	
Liveness	HEALINGSO = registerServiceDirectory ((ReceiveInfRequest.ProccessInfRequest.AnswerInfRequest) | (HealingSO) | (AskForInformation) | (ReceiveInformation) | (SendEpisodeFeedback)) $^{\omega}$
Safety	Ter acesso completo aos recursos do sistema operacional.

Detector de erros de rede

Papel: MonitorNetwork	
Descrição	Esse papel preliminar irá monitorar os estados da rede de comunicação, através da aquisição de informações em variáveis de ambiente, processos, portas etc. Irá se comunicar com papéis que requisitem informações sobre os estados do monitorados.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, MonitoringNetwork, registerServiceDirectory
Permissões	Reads OSState // *estado do S.O.* Changes OSState *// operações de alteração de elementos do S.O.*
Responsabilidades	
Liveness	MONITORNETWORK = registerServiceDirectory ((ReceiveRequest . ProccessRequest . AnswerRequest) | (MonitoringNetwork))$^{\omega}$
Safety	Ter acesso completo aos recursos do sistema operacional

Corretor de falhas de rede

Papel: HealingNetwork	
Descrição	Esse papel irá executar ações que visam recuperar o estado de elementos da rede de comunicação.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, AskForInformation, ReceiveInformation, SendEpisodeFeedback, HealingNetwork, registerServiceDirectory
Permissões	Reads KnowledgeBase // lê estratégias de correção de problemas Changes SO State *// Realiza operações no S.O.*
Responsabilidades	
Liveness	HEALINGNETWORK= registerServiceDirectory ((ReceiveInfRequest.ProccessInfRequest.AnswerInfRequest) | (HealingNetwork) | (AskForInformation) | (ReceiveInformation) | (SendEpisodeFeedback)) $^{\omega}$
Safety	Ter acesso completo aos recursos do sistema operacional.

Notificador

Papel: Notifier	
Descrição	Esse papel irá executar ações de interface entre o sistema e os usuários finais. Informações coletadas de outros papéis serão registradas em base de dados apropriada e/ou enviadas para os usuários finais.
Protocolos e Atividades	ReceiveInformation, RegisterInformation, registerServiceDirectory
Permissões	Changes LogBase // registra eventos
Responsabilidades	
Liveness	NOTIFIER = registerServiceDirectory (SendInformation | RegisterInformation | ReceiveInformation) $^{\omega}$
Safety	Ter acesso à base de registro de eventos.

Aprendiz

Papel: Learner	
Descrição	Esse papel irá executar ações de aprendizagem, registrando na base de conhecimento novos casos percebidos, registros de casos pelo usuário, coleta de feedback do usuário e acertos de casos usados.
Protocolos e Atividades	ReceiveEpisodeFeedback, // recebe feedback dos episodios de correcao RegisterFoultCase, registerServiceDirectory
Permissões	Changes KnoledgeBase // registra casos de falha // registra episódios de correção
Responsabilidades	
Liveness	LEARNER = registerServiceDirectory (ReceiveEpisodeFeedback) $^{\omega}$
Safety	Ter acesso à base de conhecimento

Apêndice C – Modelos de Interações Preliminares

Apresentamos a seguir as interações preliminares identificadas durante a fase de análise de Gaia.

ReceiveInfRequest

Nome do Protocolo: **ReceiveInfRequest**		
Iniciador: Todos atuadores	Parceiro: Todos detectores	Entrada:**Requisição para informação.**
Descrição: Mensagens utilizada para extrair informações dos detectores.		Saída: **Erro ou msg com status do objeto monitorado.**

AnswerInfRequest

Nome do Protocolo: **AnswerInfRequest**		
Iniciador: Todos detectores	Parceiro: Todos atuadores	Entrada:**Informação sobre falha.**
Descrição: Mensagens utilizada para enviar informações dos detectores.		Saída: **ok, msg recebida.**

AskForInformation

Nome do Protocolo: **AskForInformation**		
Iniciador: Todos corretores	Parceiro: Todos corretores	Entrada: **tipo da informação desejada.**
Descrição: Mensagens utilizada para troca de informações entre detectores. Usada para complementação de diagnósticos.		Saída: **ok, msg registrada para envio.**

ReceiveInformation

Nome do Protocolo: **ReceiveInformation**		
Iniciador: Todos corretores	Parceiro: Todos corretores, notificador	Entrada: **Informação sobre status do SGBD, S.O. ou rede.**
Descrição: Mensagens utilizada para troca de informações entre detectores ou entre detectores e notificador. Usada para complementação de diagnósticos (detectres) ou para repassar msg para usuários finais (notificador).		Saída: **ok, msg recebida.**

ReceiveRequest

Nome do Protocolo: **ReceiveRequest**		
Iniciador: Todos atuadores	Parceiro: Todos atuadores	Entrada: **Requisição para atuação. Identificação da atuação desejada.**
Descrição: Mensagen utilizada para solicitação de atuação.		Saída: **ok, msg registrada para envio.**

AnswerRequest

Nome do Protocolo: **AnswerRequest**		
Iniciador: Todos atuadores	Parceiro: Todos atuadores	Entrada: **Informação sobre realização da tuação.**
Descrição: Mensagen aviso de status de atuação.		Saída: **ok, msg recebida.**

ReceiveEpisodeFeedback

Nome do Protocolo: **ReceiveEpisodeFeedback**		
Iniciador: Todos corretores	Parceiro: Aprendiz	Entrada: **Informação sobre efetividade de episódio de atuação.**
Descrição: Mensagens utilizada para enriquecimento da base de conhecimento sobre resultados das atuações.		Saída: **ok, msg recebida.**

Apêndice D – Papéis definitivos

Monitor de estruturas de controle de SGBD

Papel: MonitorControlDB	
Descrição	Esse papel irá: - monitorar constantemente os estados das estruturas e controle do banco de dados alvo, monitorando processos do SGBD no S.O., em arquivos de logs (traces) de SGBD ou realizando consultas em dicionários de dados. Caso solicitado, ele irá se comunicar com outros papéis que requisitem informações, passando dados sobre o estado atual da estruturas monitoradas do banco. - executar ações que visam recuperar o estado das estruturas de controle do banco de dados e ações de correção de falhas. - notificar papel coordenador sobre falhas, ações e resultados.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, QueryKB // consulta base de conhecimento Notify, AskForInformation, ReceiveInformation, HealingDB, MonitoringDB, registerServiceDirectory
Permissões	Reads DBState *// O estado do banco de dados* Reads OSState *// O estado dos processo do SO* Reads KnowlodgeBase // lê estratégias de correção de problemas Changes DBState *// Realiza operações no SGBD*
Responsabilidades	
Liveness	MONITORCONTROLDB = registerServiceDirectory.((ReceiveInfRequest . ProccessInfRequest . AnswerInfRequest) | (MonitoringDB) || (HealingDB) | (AskForInformation) | (ReceiveInformation) | QueryKB | Notify) [ω]
Safety	Ter acesso completo ao servidor, incluindo aos processos que estão sendo executados pelo SO, processos do SGBD e ao SGBD alvo. Ter acesso à base local de conhecimento.

Monitor de estruturas físicas de SGBD

Papel: MonitorPhysicalDB

Descrição	Esse papel irá: - monitorar constantemente os estados das estruturas físicas do banco de dados alvo em arquivos de logs (traces) de SGBD, realizando consultas em dicionários de dados e verificando estados de estruturas de memória em disco (via S.O.). Caso solicitado, ele irá se comunicar com outros papéis que requisitem informações, passando dados sobre o estado atual do banco. - Esse papel irá executar ações que visam recuperar o estado do banco de dados e ações de correção de falhas. Eventualmente, também poderá requisitar informações de outros agentes. - notificar papel coordenador sobre falhas, ações e resultados.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, AskForInformation, ReceiveInformation, QueryKB // consulta base de conhecimento Notify, HealingDB, MonitoringDB, registerServiceDirectory
Permissões	Reads DBState // *O estado do banco de dados* Reads OSState // *O estado dos processo e áreas do SO* Reads KnowledgeBase // lê estratégias de correção de problemas Changes DBState // *Realiza operações no SGBD*
Responsabilidades	
Liveness	MONITORPHYSICALDB = registerServiceDirectory.((ReceiveInfRequest . ProccessInfRequest . AnswerInfRequest) | (MonitoringDB) | | (HealingDB) | (AskForInformation) | (ReceiveInformation) | QueryKB | Notify) $^{\omega}$
Safety	Ter acesso completo ao servidor, incluindo aos processos que estão sendo executados pelo SO, áreas de memória em disco e ao SGBD alvo Ter acesso à base local de conhecimento.

Monitor de estruturas lógicas de SGBD

Papel: MonitorLogicalDB

Descrição	Esse papel preliminar irá: - monitorar constantemente os estados das estruturas lógicas do banco de dados alvo em arquivos de logs (traces) de SGBD ou realizando consultas em dicionários de dados. Caso solicitado, ele irá se comunicar com outros papéis que requisitem informações, passando dados sobre o estado atual do banco. Eventualmente, também poderá requisitar informações de outros agentes. - executar ações que visam recuperar o estado do banco de dados e ações de correção de falhas. Eventualmente, também poderá requisitar informações de outros agentes. - notificar papel coordenador sobre falhas, ações e resultados.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, AskForInformation, ReceiveInformation, QueryKB // consulta base de conhecimento Notify, HealingDB, MonitoringDB, registerServiceDirectory
Permissões	Reads DBState // *O estado do banco de dados* Reads OSState // *O estado dos processos e áreas do SO* Reads KnowledgeBase // lê estratégias de correção de problemas Changes DBState // *Realiza operações no SGBD*
Responsabilidades	
Liveness	MONITORLOGICALDB = registerServiceDirectory.((ReceiveInfRequest . ProccessInfRequest . AnswerInfRequest) | (MonitoringDB) | HealingDB) | (AskForInformation) | (ReceiveInformation) | QueryKB |Notify) $^{\omega}$
Safety	Ter acesso completo ao servidor, incluindo aos processos que estão sendo executados pelo SO, áreas de memória e dicionário de dados do SGBD alvo. Ter acesso à base local de conhecimento

Monitor de SO

Papel: MonitorSO	
Descrição	Esse papel preliminar irá: - monitorar os estados do Sistema Operacional, através da aquisição de informações em variáveis de ambiente, processos e áreas de armazenamento. Irá se comunicar com papéis que requisitem informações sobre os estados monitorados. - executar ações que visam recuperar o estado de elementos do S.O. - notificar papel coordenador sobre falhas, ações e resultados.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, AskForInformation, ReceiveInformation, QueryKB // consulta base de conhecimento Notify, HealingOS, MonitoringOS, registerServiceDirectory
Permissões	Reads OSState // *estado do S.O.* Changes OSState // *operações de alteração de elementos do S.O.* Reads KnowlodgeBase // lê estratégias de correção de problemas.
Responsabilidades	
Liveness	MONITOROS = registerServiceDirectory ((ReceiveRequest . ProccessInfRequest .AnswerRequest) | (MonitoringSO) | (HealingSO) | |(AskForInformation) | (ReceiveInformation)) | QueryKB | Notify)$^{\omega}$
Safety	Ter acesso completo aos recursos do sistema operacional

Monitor de rede

Papel: MonitorNetwork	
Descrição	Esse papel preliminar irá: - monitorar os estados da rede de comunicação, através da aquisição de informações em variáveis de ambiente, processos, portas etc. Irá se comunicar com papéis que requisitem informações sobre os estados do monitorados. - executar ações que visam recuperar o estado de elementos da rede de comunicação. - notificar papel coordenador sobre falhas, ações e resultados.
Protocolos e Atividades	ReceiveInfRequest, ProccessInfRequest, AnswerInfRequest, QueryKB // consulta base de conhecimento Notify, AskForInformation, ReceiveInformation, HealingNetwork, MonitoringNetwork, registerServiceDirectory
Permissões	Reads OSState // *estado do S.O.* Changes OSState // *operações de alteração de elementos do S.O.* Reads KnowlodgeBase // lê estratégias de correção de problemas
Responsabilidades	
Liveness	MONITORNETWORK = registerServiceDirectory ((ReceiveRequest . ProccessInfRequest . AnswerRequest) | (MonitoringNetwork) | (HealingNetwork) | (AskForInformation) | (ReceiveInformation) | QueryKB | Notify)$^{\omega}$
Safety	Ter acesso completo aos recursos do sistema operacional

Notificador

Papel: Notifier	
Descrição	Esse papel irá executar ações de interface entre o sistema e os usuários finais. Informações coletadas de outros papéis serão registradas em base de dados apropriada e/ou enviadas para os usuários finais.
Protocolos e Atividades	SendInformation, ReceiveInformation, RegisterInformation, registerServiceDirectory
Permissões	Changes LogBase // registra eventos
Responsabilidades	
Liveness	NOTIFIER = registerServiceDirectory (SendInformation | RegisterInformation | ReceiveInformation) $^{\omega}$
Safety	Ter acesso à base de registro de eventos.

Gerente de Conhecimento

Papel: KnowledgeManager	
Descrição	Esse papel irá executar ações de aprendizagem, registrando na base de conhecimento novos casos percebidos, registros de casos pelo usuário, coleta de feedback do usuário e acertos de casos usados, repassados pelo papel coordenador e também deverá inferir conhecimento através de similaridade.
Protocolos e Atividades	ReceiveEpisodeFeedback, // recebe feedback dos episódios de correção RegisterFoultCase, InferCaseSolution, registerServiceDirectory
Permissões	Changes KnoledgeBase // registra casos de falha // registra episódios de correção
Responsabilidades	
Liveness	KNOWLEDGEMANAGER = registerServiceDirectory (ReceiveEpisodeFeedback) | InferCaseSolution | RegisterFoultCase) ω
Safety	Ter acesso à base de conhecimento.

Coordenador

Papel: Coordinator	
Descrição	Esse papel irá coordenar a ação dos outros papéis, elaborando planos globais de resolução de falhas e a participação e seqüência de cada papel (planos parciais). Além disso, ele se responsabilizará pelo seguimento de normas organizacionais e regras de negócio (contexto), que comporão restrições às resoluções dos planos (e.g. restrições de horário). O papel também se responsabilizará pela orientação dos papéis Notificador e Gerente de Conhecimento, os autorizando a realizar modificações nas suas respectivas bases de dados.
Protocolos e Atividades	SendInformation, ReceiveInformation, ReceiveEpisodeFeedback, // recebe feedback dos episodios de correcao Notify, RegisterServiceDirectory, CreatePlan,
Permissões	Reads KnoledgeBase // registra casos de falha // registra episódios de correção
Responsabilidades	
Liveness	COORDINATOR = RegisterServiceDirectory (ReceiveEpisodeFeedback | CreatePlan | SendInformation | ReceiveInformation | Notify) ω
Safety	Ter acesso à base de conhecimento

Apêndice E – Modelo de Serviços

Segue abaixo a lista de modelos de serviços utilizados na nossa implementação.

Serviço	**Resolução de episódio de falha**	**Registro de feedback**
Entradas	Identificação do plano global	Identificação do tipo de falha e medida de eficiência do episódio
Saídas	Status de realização concluída / Medida de efetividade	Status de realização concluída
Pré-condições	Autorização do coordenador	Feedback coletado
Pós-condições	Feedback coletado	-

Serviço	**Registro de Execução**	**Notificação de Alerta**
Entradas	Identificação do tipo de falha, agente atuador e dados da atuação (horário, plano globAL, etc)	Identificação do tipo de falha.
Saídas	Status de registro efetuado	Msg formatada para envio ao usuário.
Pré-condições	Execução concluída	Falha sem atuação automática cadastrada.
Pós-condições	-	-

Serviço	**Elaboração de Plano Global**	**Recuperação de plano parcial**
Entradas	Informações sobre tipo de falha, plano local de resolução, Regras de negócio e políticas organizacionais.	Informações sobre tipo de falha,
Saídas	Plano global de resolução;	Plano parcial de execução / indicação de falha apenas para alerta;
Pré-condições	Identificação de falha;	Identificação de falha;
Pós-condições	Registro do plano Global.	-

Datas e assinaturas

10 de outubro de 2006.

_____

Patrícia Cabral de Azevedo Restelli Tedesco

(Orientadora)

_____

Davi Lyuma Anabuki

(Orientando)