# 4
# Estudos de Caso

Os estudos experimentais [51] são uma maneira eficaz de fornecer comprovações empíricas de prova de conceitos que podem melhorar a compreensão das técnicas e métodos utilizados neste trabalho. Este trabalho de pesquisa envolveu cinco estudos de casos, usados para avaliar os elementos da abordagem proposta: Expert Committee (EC), Academic.Info (AI), Academic.Net (AN), Therapp (TH) [22, 32] e MobileInvest (MI). Os estudos de casos procuraram exercitar as propriedades do *framework*, os serviços de infraestrutura, o suporte ao projeto detalhado dos agentes e a interoperabilidade da plataforma com aplicações externas, via serviços Web.

Os cinco estudos de caso foram desenvolvidos para diferentes domínios de aplicação, tais como para a área médica, para a área de negócios, para a área acadêmica e informações científicas. Esta diversidade de domínios trouxe benefícios no processo de evolução do *framework*. Dois dos estudos de caso (EC e TH) utilizam agentes híbridos [38] (Seção 2.1.1), e envolveram um processo de desenvolvimento de complexidade média. Particularmente, o sistema EC, que simula o comportamento real de um SMA para o gerenciamento de conferências adota o conceito de agentes especialistas [38], que utilizam o *blackboard* para se comunicarem e para manter a estrutura de dados da solução.

A Tabela 2 a seguir ilustra os protótipos implementados, identificando o seu nome, o domínio de aplicação, os tipos de agente utilizados e os recursos manipulados. O sistema AN não possui recursos registrados na plataforma; ele simula um sistema externo que acessa serviços Web publicados pela plataforma pelo sistema EC. Os recursos manipulados pelas aplicações foram distribuídos através das camadas de apresentação, lógica e dados. As entidades localizadas nas camadas de apresentação e de dados são consideradas externas, isto é, não pertencem propriamente ao *framework*. Os componentes das camadas de apresentação e dados foram desenvolvidos com o auxílio de *frameworks* especialistas, como o Hibernate para dados e o Struts para apresentação. Alguns

protótipos, tais como o AI, foram simulados para prover somente elementos da camada de lógica. Para fins de registro na plataforma, somente os elementos da camada de lógica são considerados recursos.

| Nome do Sistema | Domínio | Tipos de Agente | Recursos | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | Apresentação | | Dados | Lógica | | | | |
| | | | | | | Componentes | | Agentes | Serviços | |
| | | | JSP | Serv | Beans | Nativos | WS | | Nativos | WS |
| Expert Committee | Gerenciamento Conferências | Híbrido | 16 | 16 | 12 | 4 | | 6 | 35 | 3 |
| Academic.Info | Informações Acadêmicas | — | - | - | 15 | 6 | 3 | - | 14 | |
| Academic.Net | Pesquisa Científica | — | - | - | - | - | - | - | - | - |
| Therapp | Area de Terapia Médica | Híbrido | 12 | 12 | 8 | 4 | - | 4 | 12 | - |
| Mobile Invest | Área de Negócios | Reativo | 2 | 2 | 9 | 3 | | 4 | 6 | 5 |

**Tabela 2 -** Protótipos implementados e as suas características

Os *agentes* caracterizam entidades que atendem às propriedades fundamentais de agência (Seção 2.1.1). Os *componentes* são entidades reativas que podem ser de dois tipos: *nativos* e *sw*. Os *componentes nativos* são aqueles que encapsulam funcionalidades específicas do domínio da aplicação, como objetos de acesso a dados, cujos serviços são utilizados pelos agentes. Os *componentes ws* representam serviços de sistemas externos encapsulados como serviços Web, que podem ser requisitados por agentes e componentes. Os *serviços* representam funcionalidades da camada lógica onde se localizam agentes e/ou componentes. Os elementos típicos utilizados por um SMA implementado na plataforma estão bem caracterizados no sistema EC, que utiliza todos os recursos apresentados na Tabela 1.

O primeiro estudo de caso Expert Committee (EC) é um SMA aberto, desenvolvido para dar suporte ao gerenciamento de submissões e revisões de artigos a uma conferência ou *workshop*. Ele procura simular o comportamento de agentes autônomos e adaptativos que utilizam o *blackboard* como uma área comum para compartilhamento de dados e um meio alternativo de comunicação. Através do *blackboard*, os agentes podem se adaptar a cada fase da resolução do problema, e podem alterar o estado das variáveis globais e modificar o ambiente. O EC interage com outros containers requisitando serviços remotos e provendo serviços Web. No EC foram exercitadas quase todas as funcionalidades e propriedades do *framework*, particularmente as seguintes:

- O modelo de comunicação remota da plataforma;
- O modelo de comunicação assíncrono
- A capacidade de configuração dinâmica
  - dos agentes intermediários
  - dos agentes de aplicações
- A capacidade de gerar de forma automática as especificações WSDL e UDDI para serviços Web;
- A capacidade de reutilizar serviços distribuídos;
- Os mecanismos de gerenciamento providos pelo *framework*.

O segundo protótipo Academic.Info (AI) simula uma aplicação para o gerenciamento de informações acadêmicas utilizados por um curso de pós-graduação. Composto apenas por componentes reativos, o AI procura exercitar as funcionalidades de um Component Container na plataforma. Ele provê informações como dados de instituições, de pesquisadores, de professores e de alunos. Ela mantém uma estrutura de dados que segue os padrões Lattes, dividindo as áreas e subáreas de conhecimento. O AI requisita serviços Web de um sistema externo, simulado para fornecer informações de instituições e pesquisadores da plataforma Lattes. Esforços para integrar serviços Web à plataforma Lattes[1] podem ser encontrados em [31, 74]. O AI procura exercitar particularmente as seguintes funcionalidades e propriedades:

- O funcionamento de um Component Container
- Encapsulamento de serviços Web em componentes
- Interoperabilidade com uma aplicação externa via serviços Web
- Integração da plataforma com o framework Hibernate

O terceiro protótipo Academic.Net (AN) simula o comportamento de um portal para pesquisa científica na Web. Através do portal, podem ser obtidas informações sobre conferências passadas, tais como artigos por autor, por título, por palavra-chave e por instituição. O AN utiliza serviços Web gerados e providos pelo sistema EC, acessando-os através do registro UDDI da plataforma. O

[1] Framework InterLattes (UFSC) desenvolve um trabalho para disponibilizar informações da plataforma Lattes como serviços Web.: ftp.cnpq.br/pub/doc/plataformalattes/**interlattes**/CursoFrameworkInterLattes.pdf

propósito principal deste protótipo foi testar a interoperabilidade de fora para a plataforma, isto é, uma aplicação externa requisitando via serviços Web uma funcionalidade provida por um sistema localizado na plataforma.

O quarto estudo de caso Therapp (TH) é um SMA composto por agentes híbridos que provê serviços para a área médica em terapia comportamental utilizando Realidade Virtual (RV). Os agentes asseguram ao sistema um ambiente de execução seguro de experimentação [102], onde o paciento pode ser exposto a situações que ele tem dificuldade de confrontar na vida real. Os agentes manipulam cenários de RV, e capturam os eventos acionados pelo paciente. Therapp foi o primeiro protótipo implementado na plataforma. Durante este período, nem todas as funcionalidades do *framework* estavam concluídas. Os principais propósitos foram:

- Exercitar as particularidades envolvidas no paradigma SOA: mensagens, serviços, recursos
  - As descrições das interfaces e representações dos recursos
  - A estrutura mantida em XML
  - A estrutura mantida em memória *cache*
- Exercitar propriedades não funcionais da arquitetura:
  - Flexibilidade das aplicações: via criação dinâmica de instâncias
  - Estrutura fracamente acoplada
- Exercitar os mecanismos existentes na época para facilitar o projeto detalhado dos agentes:
  - Componentes *wrapper* para encapsular objetos de acesso a dados
  - Facilidades de manipular dados em SGBDs providas pelo agente DBPool

O quinto protótipo Mobile.Invest (MI) foi projetado para prover informações a investidores da bolsa de valores. Através do sistema, o investidor pode selecionar ações, formar carteira e obter resultados *on-line* sobre o desempenho da sua carteira e das demais ações da bolsa de valores. O sistema captura as informações disponibilizadas publicamente pela BOVESPA. Um dos

atrativos do sistema é que o investidor pode obter as informações através de dispositivos móveis, tais como celular, *palm-top* e outros. Os principais propósitos deste estudo de caso foram:

- Manipular as especificações WSDL, UDDI e SOAP
- Verificar o acesso remoto à plataforma via serviços Web
- Testar o acesso à plataforma através de dispositivos móveis.

## 4.1 O Sistema Expert Committee

O sistema EC oferece suporte a diferentes atividades nas tarefas de gerenciamento de conferências e *workshops*: o envio de artigos, a atribuição de um artigo a um revisor, a seleção de revisores, a notificação da aceitação e recusa de artigos, entre outros. Agentes de software foram introduzidos no sistema EC para assistir os pesquisadores e a comissão de organização de um evento em tarefas que fazem parte do processo de revisão e submissão e que podem ser automatizadas. Os agentes utilizados no sistema EC representam o evento, os autores dos artigos, o *chair* do evento, os membros do comitê de programa (CP), o coordenador geral e os revisores.

### 4.1.1 Metodologia

A plataforma não pressupõe a utilização de nenhuma metodologia específica. A metodologia Gaia [114] foi utilizada para modelar o sistema EC, adaptada ao contexto do *framework*. O nome e a descrição de algumas fases da metodologia, tais como *Interaction Model* e *Acquiaintances*, foram adaptados para o contexto SOA da nossa abordagem e substituídos por outros dois modelos chamados *Behavior Model* e *Resource Model*, respectivamente.

A Figura 39 mostra os principais conceitos definidos pela metodologia adotada para o desenvolvimento do EC, e os relacionamentos entre os conceitos. O processo de desenvolvimento está dividido em três fases: *Requirements Statement*, *Analysis* e *Design*. Seguindo o mesmo princípio da metodologia Gaia, consideramos a fase de captura dos requisitos como sendo independente do paradigma usado nas fases de análise e projeto. No sistema EC, o modelo baseado em Cenários [70] foi utilizado para a declaração de requisitos. Os cenários

fornecem uma especificação completa para gerar documentos de requisitos em linguagem semi-estruturada, explicitando os atores, os serviços (considerados ‘episódios’), os recursos, os objetivos e o contexto que compõem o ambiente de execução. Entretanto, outros modelos para especificação de requisitos, tais como *goal-driven* [28], *task-oriented* [35] podem ser usados.

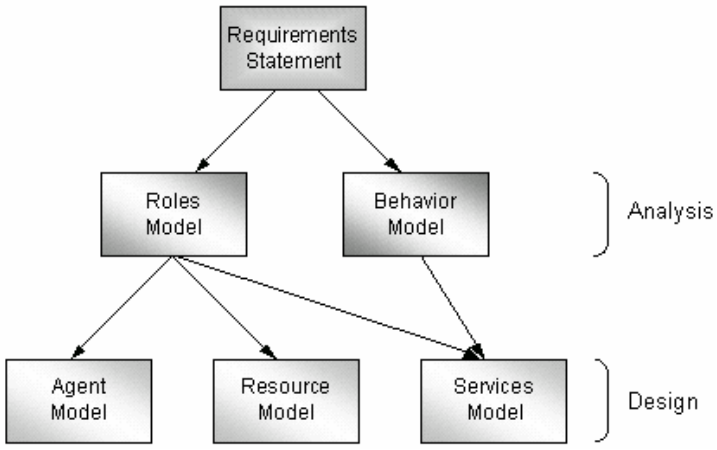


**Figura 39 -** Relacionamentos entre os modelos

Durante a fase de análise, o desenvolvimento focaliza a criação de dois modelos: Roles Model e Behavior Model. O Roles Model é um modelo estático, baseado na teoria dos papéis [11] que define as responsabilidades e os direitos associados a cada agente. Responsabilidades podem ser vistas como serviços e direitos podem ser vistos como recursos que o agente pode dispor para cumprir suas responsabilidades. Um recurso pode representar uma informação armazenada em um banco de dados ou um serviço provido por um componente ou por um outro agente.

Behavior Model é uma representação dinâmica do comportamento de um agente. Sendo uma arquitetura orientada a serviços e fracamente acoplada, não existem interações diretas entre agentes; a comunicação ocorre de forma indireta, através da arquitetura. O Diagrama de Atividades UML [12] foi utilizado para modelar o comportamento dos agentes na fase de análise, abstraindo as interações com a arquitetura. No diagrama, cada atividade corresponde a uma responsabilidade que deve ser desempenhada pelo agente.

Durante a fase de projeto, as especificações abstratas construídas na fase de análise são convertidas para conceitos mais concretos. O diagrama de atividades é

recursivamente decomposto: cada atividade identificada no diagrama de alto nível terá seu próprio diagrama de atividades em um nível mais baixo de detalhamento. No nível mais baixo de abstração, a decomposição resulta em um algoritmo bem próximo do código.

Agent Model representa o agente concreto que irá desempenhar o papel definido no Roles Model. Agent Model representa a classe abstrata Agent da plataforma estendida como classe concreta. A classe concreta herda alguns métodos *frozen* da classe abstrata e outros abstratos que são os *hot-spots* providos para implementar os agentes concretos. Os agentes concretos são implementados baseando-se nas responsabilidades e direitos do modelo de papéis.

Services Model identifica os serviços associados com cada papel, e especifica a descrição da interface desses serviços. A interface compreende os dados de entrada, os dados de retorno, o identificador e o nome amigável do serviço. Note que por definição, cada papel deverá estar associado à pelo menos um serviço. Nós entendemos serviço como uma função de um agente, ou um bloco coerente de atividades na qual um agente está engajado, similar ao conceito usado na metodologia Gaia [114].

Resource Model é derivado do modelo de papéis e pode representar serviços providos por outros agentes e componentes, ou fontes de dados que os agentes estão habilitados a acessar. Cada recurso tem uma URL associada, que é a do container onde ele se encontra. Tem certa semelhança com o modelo denominado Acquaintances da metodologia Gaia, considerando as características de uma arquitetura soa. SOA.

Os agentes do sistema EC trabalham de forma similar a especialistas em suas áreas. Existe um especialista que coordena o ciclo de vida do sistema, outro especialista em revisões, outro em resolução de conflitos, e assim por diante. Eles desenvolvem as suas atividades de maneira autônoma e se adaptam às mudanças ambientais.

O agente Event é o responsável pela criação de um novo evento (*workshop*, conferencia, etc.), pela identificação dos membros do comitê, do *chair* e dos coordenadores do evento. Quando um evento é criado, são definidos dados tais como, patrocinadores, locais, datas, e os *dead-lines* de cada fase do sistema: submissão de artigos, revisão, e envio do *camera-ready*. As datas das fases são

mantidas em memória compartilhada e controladas pelo agente Event, que pode efetuar alterações nas datas e informar os grupos via *blackboard.*

O agente Author envia um ou mais artigos para o evento e espera a resposta da revisão. No processo de submissão do artigo, o autor deve informar título do artigo, abstract, autores e instituições e a(s) área(s) na(s) qual (quais) o artigo se classifica. Caso o artigo seja aceito, o autor envia o camera-ready do artigo, i.e., a versão final do artigo.

As principais atribuições do agente Chair são selecionar os revisores, verificar se existem conflitos e enviar o resultado das revisões para os autores. Quando um novo evento é criado, o Chair distribui os artigos para os revisores de acordo com a área de interesse dos revisores e com a(s) área(s) na(s) qual (quais) o artigo se encontra. Após a revisão, os revisores a enviam o resultado para o Chair. Ele analisa as avaliações, verifica se existe algum conflito e gera o resultado final, enviando juntamente com as revisões para os autores. Na fase final do ciclo de vida do sistema, o Chair coordena o envio dos *câmera-ready* dos artigos aceitos.

O agente Reviewer tem por atribuição revisar três dos artigos submetidos ao evento, e que lhe foram atribuídos pelo Chair. Quando o revisor recebe a lista de artigos que deve revisar, ele verifica se pode ou não aceitar a proposta de acordo com os compromissos em sua agenda, com o deadline para envio de revisões e com suas áreas de interesse. Após concluir a revisão de cada artigo, o revisor envia uma mensagem para o *blackboard* informando a remessa das revisões, juntamente com o seu parecer final sobre o artigo.

O objetivo do agente Coordinator é fornecer e manter a lista de revisores de um evento. Ele monitora através do *blackboard* se existem requisições do agente Chair solicitando novos revisores. Quando o Chair pede novos revisores, ele informa o número de revisores que deseja e a área de interesse dos novos revisores. O coordenador consulta sua base de dados e fornece no mínimo 1,5 vezes o número de revisores solicitados.

O agente MCommittee representa os membros do comitê. A sua principal função é avaliar conflitos existentes nas revisões dos artigos. Um conflito ocorre quando um artigo é fortemente aprovado por um revisor e é fortemente reprovado por um outro revisor. Durante o seu ciclo de vida, ele fica monitorando o *blackboard* para verificar se existe alguma mensagem relacionada com conflitos

em artigos. Ele então captura os artigos onde ocorreram conflitos e suas respectivas revisões. Com base nas revisões, cada um dos membros vota pela aceitação ou rejeição do artigo.

Cada artigo deve ser revisto por no mínimo três revisores e cada revisor revê no máximo três artigos. Além disso, revisores não podem revisar artigos de autores da mesma instituição à qual ele está associado. Antes de enviar os artigos para os revisores, o Chair envia uma proposta de revisão; esta proposta contém uma lista de títulos dos artigos e os abstracts. O revisor avalia a proposta e informa ao Chair se aceita ou não revisar os artigos.

### 4.1.2 Especificação de Requisitos

Cenários modelam situações do mundo real e utilizam linguagem natural semi-estruturada [70]. O sistema EC possui 13 cenários[2]:

1. CE01 – Criação de um Novo Evento
2. CE02 – Submissão de Artigo
3. CE03 – Atribuição Básica de Artigos
4. CE04 – Avaliação da Proposta de Revisão
5. CE05 – Contratação de Novos Revisores
6. CE06 – Atribuição Secundária de Artigos
7. CE07 – Distribuição de Artigos aos Revisores
8. CE08 – Enviar Revisão de Artigo
9. CE09 – Avaliação das Revisões e Identificação de Conflitos
10. CE10 – Resolução de Conflitos
11. CE11 – Divulgação da Revisão de Artigos
12. CE12 – Envio do *Camera-ready*
13. CE13 – Encerramento do Evento

O ciclo de vida do sistema de EC se desenvolve através de cinco fases bem delineadas, dentro das quais ocorrem certas funcionalidades. A transição destas fases acontece apenas em uma direção, ou seja, uma vez que o sistema muda de fase, não há volta para a fase anterior. A passagem de um estado a outro é sempre

marcada por um evento temporal, chamado "*deadline*". Dentro de cada fase, podem acorrer um ou mais cenários. Os treze cenários encontram-se distribuídos dentro de cada fase da seguinte forma:

- **Fase 1 – Criação de um Novo Evento e Recepção de artigos:** o primeiro passo para iniciar um ciclo, antes de iniciar a recepção de artigos, é a criação de um novo evento. Durante esta fase, são coletados os dados do evento e definidos os *deadlines* para cada uma das fases. Esta atividade é executada pelo agente Event com o auxílio do agente humano Administrator. Após os dados do evento terem sido definidos e a *home-page* publicada, a fase de recepção de artigos propriamente dita é iniciada. Esta fase se mantém até o momento do *deadline submissão de artigos*, que leva o sistema ao próximo estado.
  - **Cenários**:
    - 01 Criação de um Novo Evento
    - 02 Submissão de Artigo
- **Fase 2 – Seleção de Revisores e Envio de Proposta:** aqui ocorre uma primeira triagem de pesquisadores que podem desempenhar o papel de revisor. Eles são selecionados pelo agente coordenador geral, que começa suas atividades capturando uma lista com as áreas dos artigos a serem revisados e a quantidade de artigos por área. De posse desta lista, o coordenador geral seleciona 1,5 vezes o número necessário de revisores, e disponibiliza os dados em uma tabela, notificando o *blackboard*. A partir daí, são enviadas propostas de revisão aos agentes revisores, que devem responder a proposta antes do *deadline alocação de revisores*.
  - **Cenários**:
    - 03 Atribuição Básica (Seleção de Revisores)
    - 04 Avaliação da Proposta de Revisão
    - 05 Contratação de Novos Revisores
- **Fase 3 – Revisão de Artigos:** o agente Chair começa esta etapa do sistema verificando se alguns dos revisores rejeitaram suas propostas. Caso isto se verifique, para efeitos de simplificação, uma nova lista de revisores será requerida ao coordenador geral, e os novos pesquisadores não terão a chance de rejeitar as novas propostas. Assim que todos os artigos já possuem 3

[2] O documento de requisitos foi elaborado em relatório técnico denominado *Expert Committee* (EC), feito pelo grupo *LES-TecCom* da PUC-Rio. Algumas inclusões foram feitas nas especificações originais.

revisores, o Chair remete os artigos para cada revisor, entrando em modo de espera pelas revisões, que podem ser enviadas até o *deadline revisão de artigos*.

- **Cenários**:
  - 06 Atribuição Secundária de Artigos
  - 07 Distribuição de Artigos aos Revisores
  - 08 Enviar Revisão dos Artigos

- **Fase 4 – Resolução de Conflitos:** de posse das revisões, o Chair analisa os artigos que possuam avaliações conflitantes, e os envia ao agente membro do comitê, que deve remeter um voto de aprovação ou rejeição. Este estado permanece até o *deadline votação de artigos*.
  - **Cenários**:
    - 09 Avaliação das Revisões e Identificação de Conflitos
    - 10 Resolução de Conflito
- **Fase 5 – Envio de Resultados/Recepção do Camera-Ready:** ao entrar nesta fase, o Chair notifica todos os autores sobre os resultados que obtiveram, aqueles que foram aprovados, devem submeter a versão *camera-ready* de seus artigos ao Chair, para serem impressos e publicados no *proceedings* conferência. O *deadline* desta fase é chamado *camera-ready*.
  - **Cenários**:
    - 11 Divulgação da Revisão de Artigos
    - 12 Envio do *Camera-Ready*
    - 13 Encerramento do Evento

Após os requisitos terem sido especificados, as responsabilidades definidas pelos cenários são mapeadas para o modelo de papéis, descrito a seguir.

### 4.1.3 Análise

Durante a fase de análise, o propósito é definir o perfil básico da organização, isto é, como os agentes cooperam para realizar os objetivos, e o que é requerido individualmente dos agentes para isto. Estas informações podem ser recuperadas através do documento de requisitos, construído durante a fase de declaração dos requisitos. O perfil da organização pode ser delimitado por dois modelos: (i) um *modelo de papéis*, que define as responsabilidades e os direitos de

um agente da aplicação e (ii) um *modelo comportamental*, que define um modelo dinâmico mostrando as atividades e trocas de mensagens entre os agentes.

O modelo de papéis tem sido utilizado por inúmeras abordagens [114, 35, 118, 99] para fornecer uma descrição abstrata de agentes de software. De acordo com a teoria dos papéis [11], um papel pode ser descrito através de dois atributos básicos: (i) *obrigações*: são as responsabilidades do papel, e indicam funcionalidade; e (ii) *permissões*: são os direitos associados com o papel, e indicam os recursos que podem ser utilizados pelos agentes. Intuitivamente, obrigações estão associadas com os serviços que um agente deve prover, enquanto as permissões estão associadas com os serviços que o agente pode dispor para cumprir suas responsabilidades.

A Tabela 3 mostra o modelo de papéis representado em um *template* dividido em duas partições: cabeçalho e corpo de detalhamento. O cabeçalho captura um conjunto de atributos de natureza geral, tais como o nome do papel, o nome da organização que ele pertence, o pacote e o endereço. O corpo de detalhamento é dividido em duas seções: *Responsabilities* e *Permissions*.



**Role Name**: Event — **Package**: mas.ecommittee
**Organization**: E-Committee — **Path**: C:\ WEB-INF\src\mas\ecommittee\committee

| Responsabilities | | | | Permissions | | | |
|---|---|---|---|---|---|---|---|
| **Service**: Create a New Event<br>**Summarized Name**: createEvent | | | | **Service**: Board.getContextAttribute<br>**Organization**: E-Committee | | | |
| **input** | **type** | **output** | **type** | **input** | **type** | **output** | **type** |
| eventData | Map | - | void | event | String | - | void |
| | | | | | | | |
| **Service**: Advanced Event Staus<br>**Summarized Name**: eventStatus | | | | **Service**: Board.setContextAttribute<br>**Organization**: E-Committee | | | |
| **input** | **type** | **output** | **type** | **input** | **type** | **output** | **type** |
| event | String | - | void | event | String | - | void |
| | | | | | | | |
| **Service**: Finalize a Event<br>**Summarized Name**: closeEvent | | | | **Service**: addUpdateEvent<br>**Organization**: E-Committee | | | |
| **input** | **type** | **output** | **type** | **input** | **type** | **output** | **type** |
| eventData | Map | - | void | eventData | Map | | |
| | | | | | | | |

**Tabela 3 -** *Template* para o papel Event

Na seção *Responsabilities* são informados os serviços que o papel provê, assim como os dados necessários para a sua execução. Na seção *Permissions* são informados os serviços que o papel pode dispor para desempenhar as suas responsabilidades. Repare que as permissões prevêem a utilização de serviços do *blackboard* para recuperar e atualizar o conteúdo de variáveis ambientais. O

modelo de papéis serve como um guia para o desenvolvedor implementar as funcionalidades dos agentes.

O Diagrama de Atividades UML [12] pode ser usado para representar o comportamento dos agentes. Ele é útil para representar os procedimentos que precisam ser implementados para preencher os *hot-spots*. Entretanto, qualquer outro modelo para especificar comportamento tais como diagrama de estados UML [33], sistemas de transição de estados [84, 116], protocolos de comportamento [93] ou outros modelos baseados em máquinas de estados [117] podem ser usados. A Figura 40 mostra o diagrama de atividades do sistema EC.

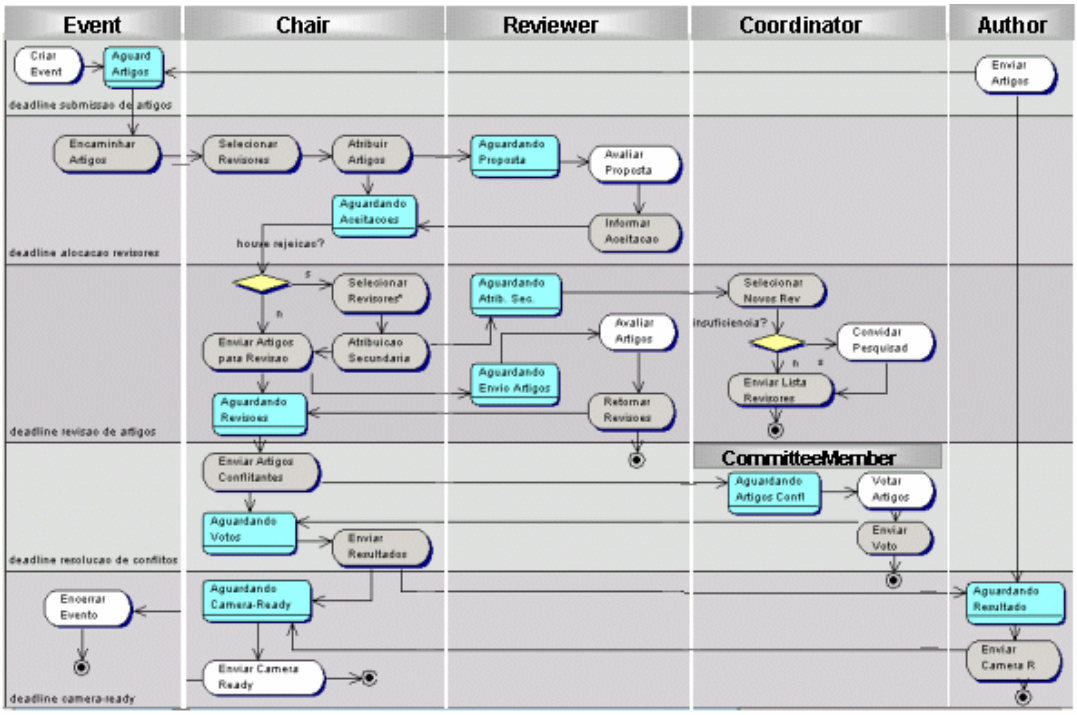


**Figura 40 -** Diagrama de atividades do sistema EC

As fases do ciclo de vida do sistema EC estão representadas pelos blocos horizontais e os *swimlanes* verticais representam o ciclo de vida dos agentes. O diagrama de atividades fornece uma visão seqüencial e temporal das atividades realizadas por cada um dos agentes, e torna fácil visualizar a implementação do *hot-spot lifecycle*, já que cada atividade do seu ciclo de vida encontra-se representada no diagrama. As atividades representadas no diagrama correspondem a serviços, que podem ser de dois tipos:

- serviços da camada lógica: atividades representadas por elipses cinza, caracterizam os procedimentos relacionados com as funcionalidades lógicas do agente;
- serviços da camada de apresentação: interfaces gráficas representados por elipses brancas identificam as JSPs e *servlets*.

O ciclo de vida dos agentes se caracteriza por constantes *loops* para verificar o estado das variáveis de ambiente. Estes *loops* caracterizam estados, representados no diagrama pelos retângulos; dependendo do estado das variáveis, o agente executa diferentes ações. Estas ações podem ser obtidas através de uma representação de mais baixo nível, onde cada serviço corresponde a um novo diagrama de estados. Procuramos abstrair estas representações, mostrando na seção a seguir detalhes da implementação dos *hot-spots* do *framework*, ilustrados com diagramas de seqüência UML.



### 4.1.4 Projeto Detalhado e Implementação

Esta seção focaliza os aspectos relacionados com o projeto detalhado e a implementação dos agentes do EC. Os aspectos foram modelados como um refinamento do diagrama de atividades apresentado na Figura 40. Essencialmente, o projeto detalhado consiste na implementação dos dois *hot-spots* principais providos pela classe abstrata. O primeiro denominado *lifecycle* implementa o *workflow* dos agentes e o segundo é a interface *provide*, por onde o agente trata as requisições de serviços que ele recebe.

#### 4.1.4.1 Os Agentes do EC

Para o gerenciamento do seu ciclo de vida, os agentes utilizam o *blackboard*, que provê um poderoso mecanismo de suporte ao modelo de comunicação dos agentes e ao *workflow* dos agentes. Ele mantém um conjunto e variáveis globais que controlam o estado do ambiente e o estado da resolução do problema. Os agentes trabalham na estrutura de dados e vão modificando os dados à medida que as fases do sistema evoluem.

O primeiro passo para o desenvolvimento é estender a classe abstrata Agent. O segundo passo é completar a implementação dos dois principais *hot-spots* providos pela classe abstrata: *lifecycle* e *provide*. A Figura 41 ilustra parcialmente a implementação do *hot-spot lifecycle*. O algoritmo do ciclo de vida se caracteriza por *loops* que consultam o *status* da variável *eventState* no *blackboard*, que contém sempre o estado corrente da fase do sistema.

```
28    @Override
29    public void lifeCycle() throws LifeCycleException
30    {
31        // Se registra como ouvinte do grupo ecommittee
32        Board.addMessageListener("ecommittee",this);
33
34        try
35        {
36            // Fase de Submissão
37            while(Board.getContextAttribute("eventState").equals("submission"))
38            {
39                Thread.sleep(2000);
40            }
41
42            // Fase de Alocação de Revisores
43            while(Board.getContextAttribute("eventState").equals("allocation"))
44            {
45                // SE os revisores ainda não foram selecionados
46                if (!reviewersSelected)
47                {
48                    selectReviewers();
49                    reviewersSelected=true;
50                }
51
52                Thread.sleep(2000);
53            }
54
55            // Fase de Revisão de Artigos
56            while(Board.getContextAttribute("eventState").equals("revision"))
57            {
58                // SE os artigos ainda não foram enviados
59                if (!articlesSent)
```

**Figura 41 -** Método lifecycle do agente Chair

A linha 32 mostra o agente se registrando como ouvinte de mensagens do *blackboard* no grupo *ecommittee*. A primeira fase do ciclo de vida do sistema é *submission*, que ocorre durante o processo de submissão de artigos. Durante esta fase, o agente Chair não executa nenhuma ação. A linha 43 mostra o agente invocando o método *getContextAtribute* do *blackboard* para capturar o estado da variável de sistema *eventState*. Quando o estado mudar para *allocation*, ele inicia o processo de seleção de revisores, invocando o método *selectReviewers* (linha 48). Note que ele verifica o conteúdo da variável *reviewerSelected* e só executa a ação se o conteúdo for falso. Após o encerramento da fase de alocação de revisores, o Chair inicia as ações da fase de revisão (*revision*). Na linha 59, ele verifica se ainda existem artigos sem revisores, e logo a seguir, na linha 62, ele verifica se houveram rejeições nas propostas de revisão. Caso algum revisor

humano rejeite a proposta de revisão, seu agente correspondente envia uma notificação via *blackboard* ao Chair, informando-o sobre a rejeição.

O segundo *hot-spot* da classe Agent que precisa ser completado pelo desenvolvedor é a interface *provide*, por onde ele atende requisições de serviços. A Figura 42 mostra um fragmento de código da interface *provide* do agente *Coordinator*. A interface possui como parâmetros o nome do serviço, um objeto do tipo Map contendo os dados da requisição e um objeto List que contém os dados que retornam após a execução do serviço. Estas especificações fazem parte da descrição do protocolo interno MIDAS e são transparentes ao desenvolvedor.

```
17 public class Coordinator extends Agent
18 {
19     @Override
20     public void provide(String service, Map in, List out) throws ServiceException
21     {
22         if (service.equals("selectReviewers"))
23         {
24             // Variáveis de Informação
25             List articleBeans;
26
27             // Referência que guarda uma requisição de serviço
28             ServiceWrapper serviceWrap = null;
29
30             try
31             {
32                 // Acessa a interface requerente para obter um objeto de requisição
33                 serviceWrap = require("ExpertCommitee","getUnrevisedArticles");
34
35                 // Dispara o serviço de forma síncrona, recuperando o resultado
36                 articleBeans = serviceWrap.run();
37
38                 // Opera a seleção dos revisores para os artigos
39                 operateSelection(articleBeans);
40
41                 // Notifica os ouvintes do grupo "ecommittee" que os revisores
42                 // foram alocados aos seus artigos.
43                 Board.writeOnBoard(1,"reviewersSelected","ecommittee",this);
44             }
```

**Figura 42 -** Implementação de interface provide do agente Coordinator

O fragmento de código mostra os diferentes procedimentos executados para o serviço *selectReviewers*. Na linha 22 ele verifica se a requisição é para o serviço *selectReviewers*, e efetua os procedimentos para a execução do serviço nas linhas seguintes. Na linha 43, após selecionar os revisores, o agente Coordinator informa ao *blackboard* que os revisores já foram selecionados.

A Figura 43 mostra este procedimento no diagrama de seqüência. Quando o Coordinator informa ao *blackboard* que os revisores já foram selecionados, a classe Control do *blackboard* captura a mudança no estado da variável global "*event*" e notifica os agentes registrados neste grupo ou todos os agentes.

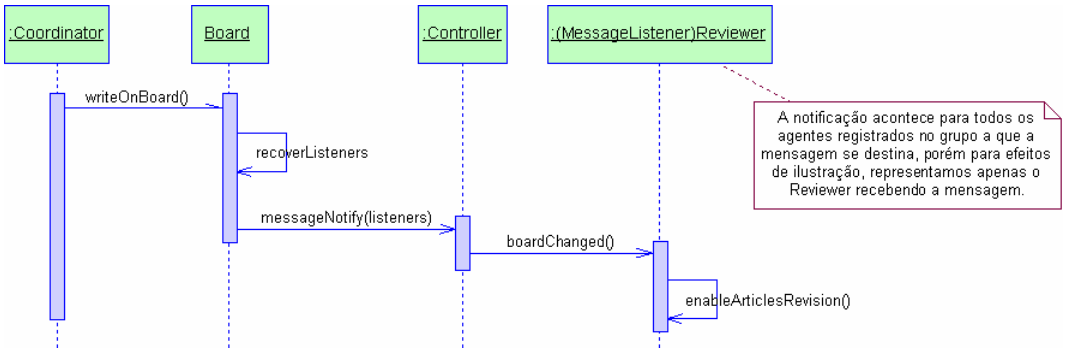


**Figura 43 -** O agente Coordinator escreve uma mensagem no blackboard

A conseqüência desta notificação está ilustrada na Figura 44, que mostra um fragmento de código do agente Reviewer, que utiliza o *blackboard* para verificar se os revisores já foram selecionados.

```
87    public void boardChanged(Message msg)
88    {
89        // SE os revisores foram devidamente selecionados
90        if (msg.getData().equals("reviewersSelected"))
91        {
92            // Habilita a entrada da revisão no sistema por
93            // parte dos revisores humanos
94            enableArticlesRevision();
95        }
96    }
```

**Figura 44 -** Implementação da interface boardChanged do agente Reviewer

O agente Reviewer, ao saber que os revisores foram selecionados, habilita a possibilidade da entrada no sistema dos resultados das avaliações efetuadas pelos revisores humanos através da interface *boardChanged*. Os agentes são notificados de mudanças no ambiente e da chegada de mensagens enviadas por outros agentes através desta interface, que funciona como um *sensor*. Na linha 90, o Reviewer verifica o conteúdo da mensagem; se ele for *reviewersSelected*, o agente invoca o método *enableHumanRevision*, avisando ao grupo de revisores que os artigos podem ser recuperados para fazer o *download* no *hiperlink* informado.

A Figura 45 mostra um fragmento de código do método *operateSelection* do agente Coordinator.

```
57    private void operateSelection(List<Articlebean> articleBeans) throws ServiceException
58    {
59        // Cache para listas já recuperadas de pesquisadors
60        HashMap<Integer,List> researchers = new HashMap<Integer,List>();
61
62        // PARA cada artigo
63        for (Articlebean article : articleBeans)
64        {
65            // Recupera o código da área de atuação do artigo
66            int articleArea = article.getArea();
67
68            // SE pesquisadores para essa área ainda não foram recuperados
69            if (!researchers.containsKey(articleArea))
70            {
71                // Utiliza a interface requerida para obter uma requisição
72                ServiceWrapper service = require("Project Management","getResearchersByArea");
73
74                // Adiciona o código da área como parâmetro
75                service.addParameter("area",articleArea);
76
77                // Dispara o serviço de forma síncrona recuperando os pesquisadores
78                List newResearchers = null;
79
80                try
81                {
82                    newResearchers = service.run();
83                }
84                catch (Exception e)
85                {
86                    throw new ServiceException("Unable to recover researchers from Project Management",e);
87                }
88
89                // Guarda os pesquisadores para a nova área no mapa de pesquisadores
90                // utilizando o código da área como índice
91                researchers.put(articleArea,newResearchers);
92            }
93        }
```

**Figura 45** - Método operateSelection do agente Coordinator

O sistema AI localizado em um container CC remoto possui um serviço denominado *getResearchersByArea* para efetuar a seleção de pesquisadores por área ou especialidade que interessa ao agente Coordinator. Para invocar o serviço, ele solicita à interface *require* o objeto de requisição *ServiceWrapper* (linha 72). Após obter o objeto, ele adiciona os dados necessários (linha 75) e invoca o serviço de forma síncrona utilizando o método *run* (linha 82). Os dados de retorno são adicionados na lista de pesquisadores, utilizando o código da área como índice (linha 91).

O diagrama de seqüência da Figura 46 ilustra o que ocorre entre as linhas 72 a 86 do fragmento de código anterior. O diagrama omite o passo que envolve a obtenção do *ServiceWrapper* (linha 72) pois este procedimento já foi ilustrado. Omite também os procedimentos tomados pelo Proxy do container remoto, pois igualmente os mesmos procedimentos já foram ilustrados no esmo local. Repare que os dois únicos procedimentos efetuados pelo agente Coordinator são adicionar os parâmetros ao objeto *wrapper* e disparar a execução do serviço invocando o método *run*.

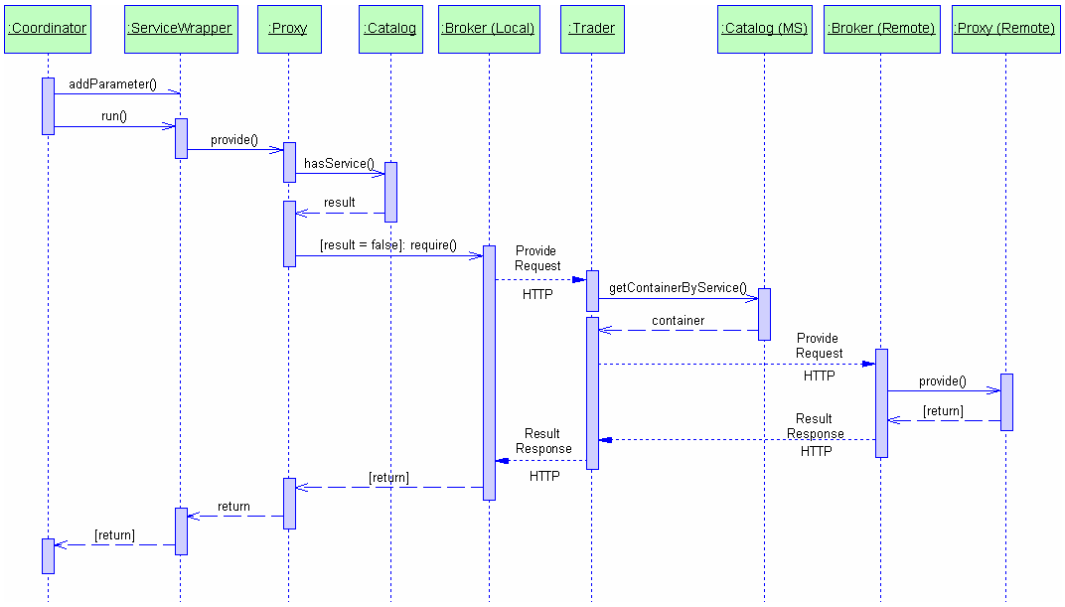


**Figura 46 -** Invocação remota se serviço efetuada pelo agente Coordinator

Através do exemplo ficou clara a maneira simples e transparente de efetuar uma requisição remota de serviço. Ela ocorre como se o serviço estivesse no próprio container. Basta adicionar os dados no *wrapper* de requisição e invocar o método *run*. A performance do sistema pode ser melhorada com a utilização de chamadas assíncronas, que escondem latência de comunicação e possibilitam melhorar a eficiência. O fragmento de código da Figura 47 mostra uma chamada assíncrona efetuada pelo agente Chair.

```
31    public void verifyAcceptedArticles() throws ServiceException
32    {
33        List<ArticleBean> articles;
34
35        try
36        {
37            // Solicitando serviço que recupera os artigos aceitos
38            ServiceWrapper wrapper = require("ExpertCommittee","getAcceptedArticles");
39
40            // Executando serviço de forma síncrona e recuperando a lista de resposta
41            articles = wrapper.run();
42
43            // PARA cada artigo
44            for (ArticleBean article : articles)
45            {
46                String      articleName = article.getName();
47                AuthorBean  author      = article.getAuthorBean();
48
49                // Solicitando o serviço que notifica via email o autor do artigo
50                ServiceWrapper wrapper2 = require("ExpertCommittee","notifyAuthor");
51
52                // Adiciona Parâmetros
53                wrapper2.addParameter("article",articleName);
54                wrapper2.addParameter("author",author);
55
56                // Disparando o serviço de forma assíncrona, já que não é necessário
57                // esperar o término da execução
58                wrapper2.submit();
59            }
60        }
```

**Figura 47 -** Método verifyAcceptedArticles do agente Chair

Durante o ciclo de vida do sistema, quando o processo de revisão de artigos é concluído, o agente Chair deve notificar os autores que tiveram artigos aceitos. Para isto, ele invoca o serviço *notifyAuthor* e obtém de sua interface requerente o objeto *wrapper* (linha 50) que encapsula a requisição. Nas linhas 53 e 54, são adicionados os dados, e a requisição é disparada de forma assíncrona utilizando o método *submitService* (linha 58). Se este serviço fosse executado de forma síncrona, poderia ser muito mais demorado, já que entre a chamada de cada um estaria o longo intervalo de se esperar a composição, emissão e envio do *e-mail*.

#### 4.1.4.2
#### Os Componentes do EC

O sistema EC utiliza vários componentes, que encapsulam funcionalidades complexas específicas do domínio. Eles desempenham importante papel, pois oferecem serviços de aceso a dados para recuperar informações normalmente distribuídas em várias tabelas. Os serviços podem ser reutilizados, e economizar grande quantidade de linhas de código e esforço de programação durante a implementação dos agentes. O EC utiliza o *framework* Hibernate para facilitar o mapeamento objeto-relacional. A Figura 48 ilustra como um agente pode requisitar um serviço de acesso a dados de um componente. Component representa a classe abstrata implementada, capaz de executar *queries* complexas utilizando os objetos *bean* do Hibernate.

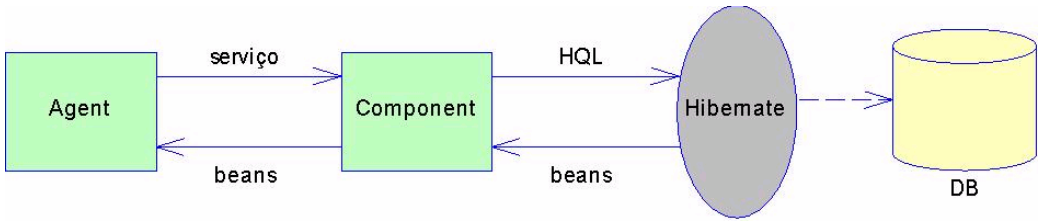


**Figura 48 -** Agente requisitando serviço de acesso a dados p/componente

A grande vantagem de utilizar o *framework* Hibernate é que as tabelas são mapeadas para arquivos XML, e podem ser acessados através de uma linguagem HQL, similar ao SQL, com grande poder para manipulação de dados. O agente DBPool provê um método denominado *sectionFactory* que fornece uma conexão para criar sessões no Hibernate. Quando um componente ou agente solicita um serviço de acesso a dados, o agente DBPool lhe fornece uma sessão já criada, que entende os comandos HQL submetidos à arquitetura. O Hibernate permite

configurar *drivers* e *Strings* de conexão em tempo de execução, e é compatível com qualquer dos bancos de dados utilizados no mercado que suportam JDBC. Quatro componentes foram implementados no sistema EC:

1. ArticleData: fornece serviços sobre os artigos, tais como artigo por nome, por autor, por instituição, por palavra chave.
2. AuthorData: provê serviços sobre o autor, tais como artigo(s) submetido(s), instituição do autor.
3. EventData: fornece serviços sobre o evento, tais como datas de *dead-line*, a variável de estado do sistema *eventState*.
4. ResearcherData: fornece informações sobre os pesquisadores, que podem desempenhar papel de revisor, chair, coordenador, autor e comitê.

A Figura 49 mostra um fragmento de código do componente ArticleData, que executa uma *query* no Hibernate para recuperar artigos sem revisores.

```
13 public class ArticleData extends Component
14 {
15     @Override
16     public void provide(String service, Map in, List out) throws ServiceException
17     {
18         if (service.equals("getUnrevisedArticles"))
19         {
20             // Objeto de Sessão do Hibernate
21             Session session = null;
22
23             // Recuperando Sessão com o Hibernate
24             try
25             {
26                 session = DBPool.getHibernateSession("ecommittee");
27             }
28             catch (Exception e)
29             {
30                 throw new ServiceException("Unable to open hibernate session for ecommittee");
31             }
32
33             // HQL que recupera os artigos que não possuem qualquer revisor alocado.
34             Query query = session.createQuery("from ecommittee.beans.ArticleBean as articleBean " +
35                                               "where count(articleBean.evalBean) = 0");
36
37             // Recuperando resultado
38             List result = query.list();
39
40             // Adicionando beans do resultado à lista de saída
41             out.addAll(result);
42         }
```

**Figura 49 -** Componente ArticleData efetuando uma query no Hibernate

Na linha 26, o agente DBPool fornece uma *session* do Hibernate, e na linha 34, uma *query* efetuada com a linguagem HQL recupera todas as instâncias de artigos que não possuem revisores. Na *query*, *evalBean* é um *bean* de relacionamento entre um artigo e seus revisores, que guarda o código do revisor, e

a nota dada por ele para aquele artigo. O resultado é atribuído à variável *result* (linha 38). Na linha 41, o resultado é atribuído ao argumento *out*, e passado por referência ao cliente.

O diagrama de seqüência na Figura 50 ilustra este procedimento, e mostra o componente ArticleData executando o serviço "*getUnrevisedArticles*" que recupera todos os *beans* de artigos não revisados.

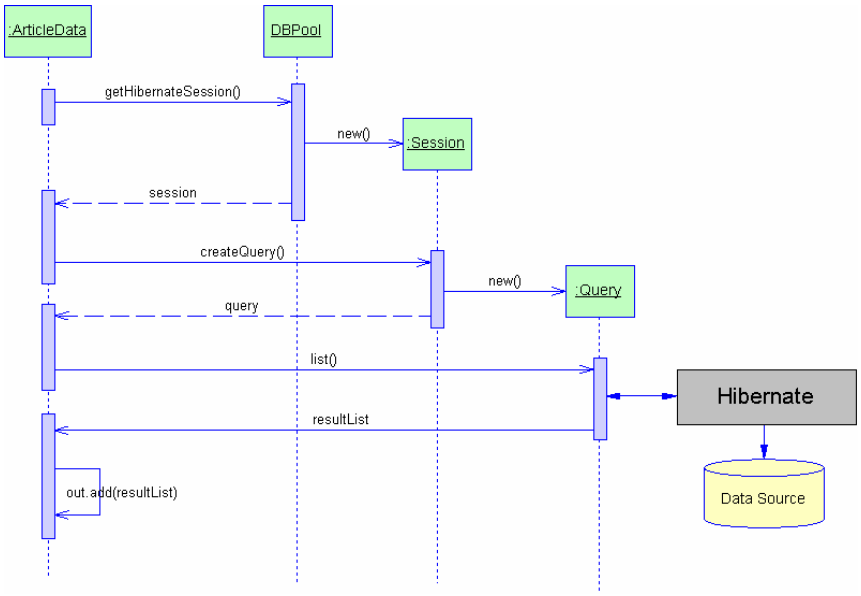


**Figura 50 -** Componente ArticleData executando getUnrevisedArticles

### 4.1.5
### As Tarefas de Gerenciamento

Durante a execução das aplicações, as atividades de gerenciamento ocorrem em duas modalidades: local e global. A modalidade de gerenciamento local envolve a manipulação dos recursos e o gerenciamento do ciclo de vida de um container e a manutenção da sua estrutura de recursos: organizações, aplicações, agentes, componentes, serviços e fontes de dados.

A modalidade de gerenciamento global é efetuada no servidor e focaliza o gerenciamento da plataforma como um todo. Quando um container se registra na plataforma, ele passa a sua estrutura de recursos para o servidor para ser consolidada na estrutura geral de recursos. O mesmo procedimento é executado sempre que ocorre uma mudança na estrutura de recursos de um container. O agente S-Manager disponibiliza um assistente GUI para auxiliar nas tarefas de gerenciamento global da plataforma, como mostra a Figura 51.

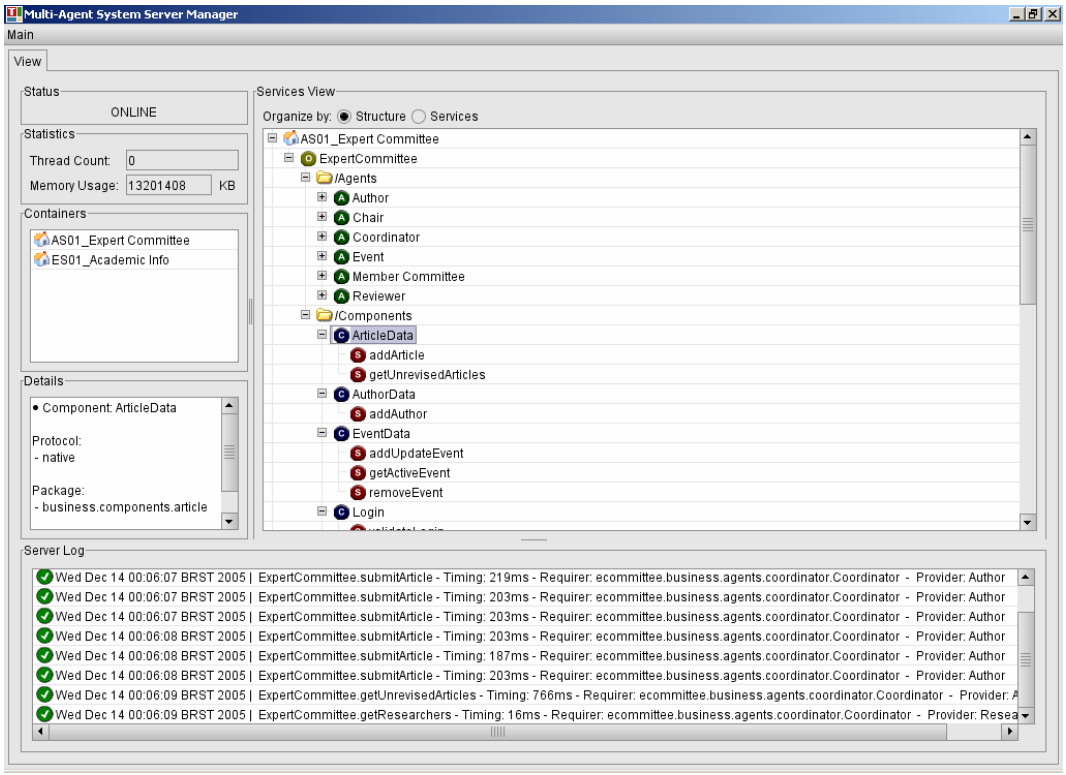


**Figura 51 -** Assistente GUI para a modalidade de gerenciamento global

No painel *Containers* do lado superior esquerdo podem ser vistos os dois containers registrados na plataforma e utilizados neste domínio: Expert Committee e AcademicInfo. O painel do lado direito mostra uma hierarquia de recursos navegável e organizada por estrutura. Ao ser selecionada uma entidade, os detalhes são exibidos no painel *Details*, do lado esquerdo da janela. Na parte inferior da janela, o painel *Server Log* mostra detalhes de todas as transações sendo executadas.

Para implementar as classes da camada de apresentação, foi utilizada a tecnologia JSP (Java Server Pages) [107] com *servlets* da Sun Microsystems. A camada de apresentação foi construída sobre o modelo de arquitetura MVC (Model-View-Controller). Os *servlets* e páginas JSP se comunicam com os atributos do objeto *httpRequest*. Através da variável *formAction* das JSPs, os dados submetido podem ser capturados e processados pelo *servlet* correspondente. Os *servlets* se comunicam com os containers para enviar dados para os agentes ou para solicitar serviços.

## 4.2
## O Protótipo Academic.Info

O protótipo AI executa em um container CC e desempenha papel importante, pois fornece informações de interesse ao EC, tais como informações sobre pesquisadores e instituições, utilizadas na seleção de revisores. Os componentes implementados no container fornecem serviços que podem ser reusados na plataforma. A estrutura de dados utilizado no AI se baseia no padrão fornecido pela plataforma Lattes, onde os pesquisadores estão classificados por grande área, área e subárea. O modelo mantém a especialidade e proficiência dos pesquisadores. A Figura 52 mostra uma vista parcial da estrutura de recursos do container AI.

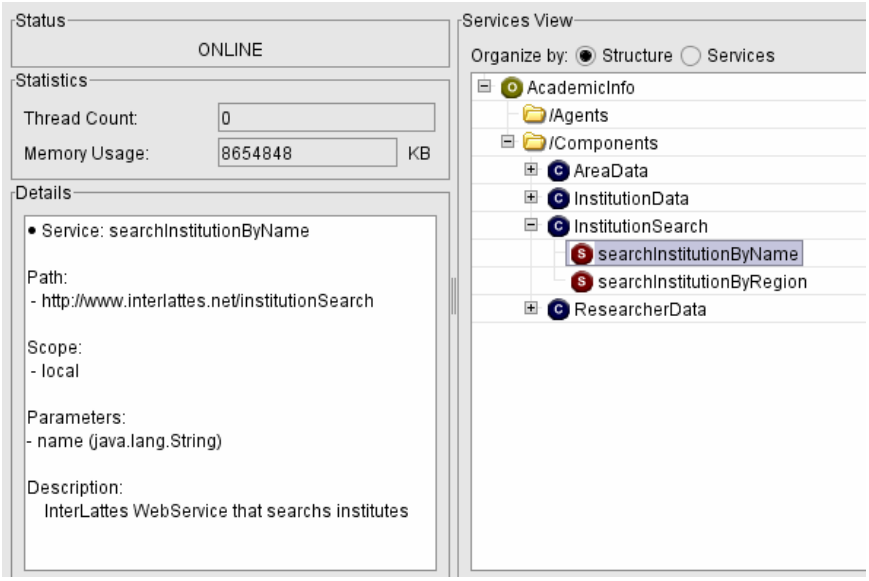


**Figura 52 -** Componente representando um serviço Web em um Container AI

O painel do lado direito da janela mostra a estrutura de recursos do container. Nela podem ser vistos os *componentes* AreaData, InstitutionData, ResearcherData, e o *componente ws* InstitutionSearch. O componente InstitutionSearch encapsula dois serviços Web: *searchInstitutionByName* e *searchInstitutionByRegion*, que localizam instituições por nome ou por região.

O painel do lado esquerdo mostra detalhes da especificação do componente InstitutionSearch. É importante notar que o tipo de protocolo definido para este serviço é SOAP. Quando uma requisição é feita a um *componente ws*, é o agente Proxy que detecta a diferença de protocolos, e redireciona a requisição ao servidor

MS (ao invés de pedir a classe Factory para criar e invocar uma instância de uma entidade nativa). Uma vez recebida no servidor pelo agente S-Broker, a requisição é encaminhada ao Adapter, que a converte numa requisição SOAP. O retorno do processamento é capturado e devolvido no List de saída. Este fluxo pode ser acompanhado no diagrama de seqüência da Figura 53.

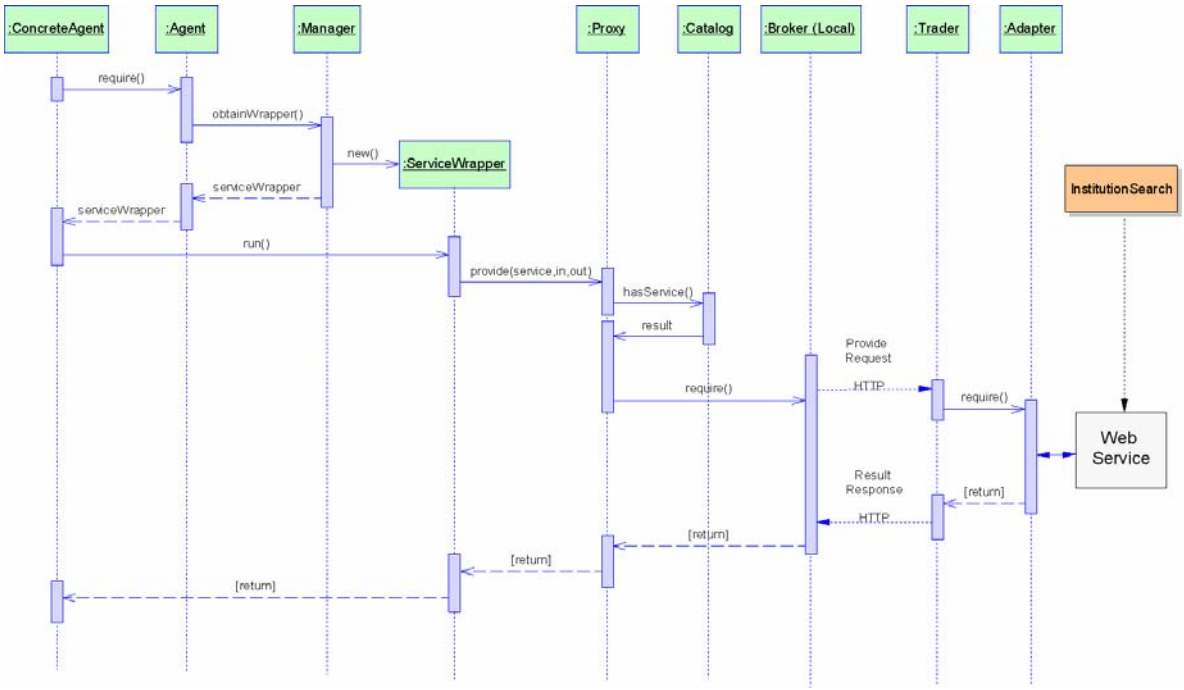


**Figura 53 -** Agente invocando serviço Web representado por componente

Além dos serviços Web externos que foram mapeados para o componente InstitutionSearch, foram implementados mais os seguintes componentes:

1. AreaData: este componente fornece informações sobre áreas, subáreas e especialidades, que servem para a classificação de pesquisadores.
2. ResearcherData: este componente fornece informações sobre pesquisadores, tais como a especialização, a área da sua linha de pesquisa, instituição a que pertence, dentre outros.
3. InstitutionData: fornece informações básica sobre a instituição, tais como o seu nome, pesquisadores vinculados e outros. Este componente encapsula uma chamada para um serviço Web, provido pelo sistema AN.

A Figura 54 mostra um fragmento de código do componente ResearcherData executando uma *query* para recuperar pesquisadores por área. No

container AI, o componente ResearcherData oferece um serviço interessante ao sistema EC, que fornece informações sobre pesquisadores, tais como os dados gerais, o currículo Lattes, as áreas e sub-áreas de atuação, suas especialidades, etc que são usadas durante o processo de seleção de revisores.

```
14 public class ResearcherData extends Component
15 {
16     @Override
17     public void provide(String service, Map in, List out) throws ServiceException
18     {
19         if (service.equals("getResearchersByArea"))
20         {
21             // Recuperando nome da área
22             String areaName = (String)in.get("areaName");
23 
24             // Objeto de Sessão do Hibernate
25             Session session = null;
26 
27             // Recuperando Sessão com o Hibernate
28             try
29             {
30                 session = DBPool.getHibernateSession("ecommittee");
31             }
32             catch (Exception e)
33             {
34                 throw new ServiceException("Unable to open hibernate session for ecommittee");
35             }
36 
37             // HQL que recupera os pesquisadores de uma certa área
38             Query query = session.createQuery("from pmanagement.beans.ResearcherBean as researcherBean," +
39                                               "where researcherBean.proficiency = "+areaName);
40 
41             // Recuperando resultado
42             List result = query.list();
43 
44             // Adicionando beans do resultado à lista de saída
45             out.addAll(result);
46         }
```

**Figura 54** - Método provide do componente ResearcherData

A sintaxe do código para a criação da sessão Hibernate e execução da *query* é similar àquela apresentada no sistema EC. O agente DBPool fornece uma sessão para conexão com o banco de dados (linha 30) e a *query* na linha 38 recupera pesquisadores de uma certa área. O resultado da *query* retorna no argumento *out*.

O componente InstitutionData oferece serviços de consulta local à instituições, da mesma forma que o componente ResearcherData fornece para pesquisadores. Para enriquecer a sua base de dados, os componentes capturam dados de um sistema externo (simulado) que provê serviços Web. Os dados são informações da plataforma Lattes relacionadas a pesquisadores e instituições. Neste sistema podem ser recuperadas informações da plataforma Lattes sobre instituições de ensino e pesquisa no Brasil (tais como nome, endereço, etc.) e informações sobre pesquisadores (área de conhecimento, subárea, especialidade).

## 4.3
## O Protótipo Academic.Net

O principal propósito do protótipo Academic.Net (AN) é demonstrar a interoperabilidade entre aplicações externas e a plataforma. O sistema EC mantém em sua estrutura de dados o histórico de eventos anteriores, e o AN utiliza serviços Web publicados pelo sistema EC. Para demonstrar a utilização destas funções como serviços Web, um portal com JSP foi construído. A aplicação implementa um mecanismo de busca para localizar artigos publicados em eventos passados por autor, por instituição e por título, como pode ser visto na Figura 55.

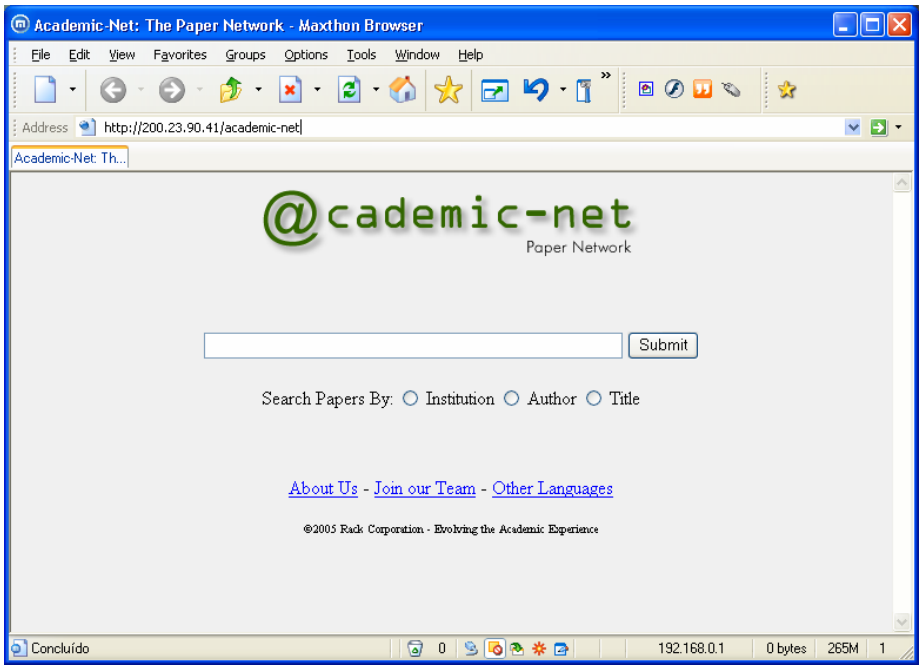


**Figura 55 -** O protótipo Academic.Net

Este protótipo desempenha outro papel nos estudos de caso. Ele simula o comportamento de um sistema externo “caixa-preta” que fornece serviços Web para a aplicação AI. Ele simula um sistema que captura informações sobre pesquisadores e instituições da plataforma Lattes. Por exemplo, ele fornece dados de instituições capturados da plataforma Lattes como um serviço Web. Da mesma forma, ele provê informações sobre listas com áreas, sub-áreas e especializações por subárea classificadas seguindo o padrão definido na plataforma Lattes. O principal propósito neste caso é demonstrar a interoperabilidade da plataforma, que utiliza o portal como um sistema externo.

## 4.4
## O Protótipo Therapp

Therapp[3] [22, 32] é uma aplicação baseada em agentes que provê um ambiente para a aplicação de terapias comportamentais utilizando Realidade Virtual (RV). As aplicações de RV proporcionam um ambiente seguro de experimentação [102], onde o paciente pode ser exposto a situações que tem dificuldade de confrontar na vida real em um ambiente controlado.

Os requisitos da aplicação Therapp foram documentados utilizando o método VORD (Viewpoint Oriented Requirements Definition) [68]. Esta técnica baseia-se na identificação de todos os pontos de vista dos utilizadores e *stakeholders* do sistema, e através destes pontos de vista obtém-se os serviços que serão prestados e as restrições que se aplicam. Para desempenhar os papéis definidos pelos requisitos, foram criados quatro agentes de software: Tutor, Aluno, Guia e Cenário. Os agentes interagem com os atores humanos e auxiliam os serviços delimitados pelos cenários.

O agente Tutor é o responsável pela condução da terapia, como aplicar cenários a alunos, registrar reações físicas do aluno e gerenciar os seus dados. O Tutor manipula cenários de RV utilizando a tecnologia de integração *External Authoring Interface* (EAI), que permite a um *applet* de Java se comunicar com um cenário de RV. O agente *Tutor* também mantém e gerencia um conjunto de informações de interesse do agente humano tutor. Ele auxilia o agente humano tabulando dados de alunos, fazendo análises e disponibilizando informações sobre a evolução do aluno nas terapias. Colaborando com o agente Guia, ele pode acessar o banco de dados de interação aluno-RV e fazer análises comportamentais baseadas em comparações com resultados obtidos nas terapias.

O agente Aluno é responsável pelos serviços relativos a alunos, tais como manter informações sobre o cadastro de alunos e acessar dados da anamnese. Estes serviços são subseqüentemente divididos em sub-serviços específicos, tais como inserir aluno, remover aluno, alterar aluno e etc.

[3] A aplicação *Therapp* foi desenvolvida como Projeto Final de Graduação do curso de Ciência da Computação, da UniverCidade (Centro Universitário do Rio de Janeiro), unidade Ipanema, do aluno Nuno Caminada, que orientei.

O agente Guia é responsável pela captura, armazenamento e gerenciamento dos dados resultantes da interação aluno-RV. Estas informações são utilizadas para delimitar o perfil de cada aluno. Quando o aluno manipula um cenário, o agente guia pode verificar aonde o indivíduo vai, com que freqüência, o que ele faz ou deixa de fazer. Estas informações são mantidas em séries históricas, o que possibilita obter padrões de comportamento para determinadas faixas do espectro autista e auxiliar qualitativamente o terapeuta na seleção e aplicação de cenários.

O agente Cenário é responsável pelo gerenciamento dos cenários de RV utilizados na terapia. Ele utiliza ferramentas de RV pode disponibilizar os cenários em diferentes dispositivos de saída, como monitor de vídeo, estação de trabalho ou capacetes de RV. Ele mantém um repositório de cenários, de diferentes formatos, organizados em uma hierarquia de tipos e classificados por nível de complexidade, que pode ser navegado pelo terapeuta.

Em uma seção de terapia, terapeuta e paciente devem estar juntos, para que o primeiro possa conduzir a sessão e observar a utilização do ambiente de RV. O sistema exibe então uma interface personalizada para aquele par paciente/terapeuta, onde estão disponíveis informações da anamnese, que possui trechos permanentes e trechos que compreendem históricos que podem ser atualizados à medida que as sessões progridem.

A aplicação reside em um container AC, e seu acesso é feito através de um *browser* comum, dotado de qualquer *plug-in* para utilização de arquivos VRML97, como Cortona[4]. O usuário administrador é o terapeuta, que pode se cadastrar incluindo seus dados e selecionando um *login* e senha, através das quais está habilitado aos serviços definidos para o seu papel.

A Figura 56 mostra um assistente GUI provido elo agente *Tutor* para o terapeuta que deseja iniciar uma seção. O *frame* do lado esquerdo da janela possibilita ao terapeuta selecionar as tarefas. Os cenários foram construídos utilizando a ferramenta de RV *Internet Space Builder*, e procuram simular situações que ocorrem na vida real, e observar as ações do aluno. O propósito deste cenário (nível de complexidade médio) é ambientar o autista com as interações e transmitir as noções de funcionamento do ambiente, como a maneira de se movimentar e o que está vendo. As seqüências de ações do aluno nos

[4] <*http://www.parallelgraphics.com*>

cenários de RV são persistidas pelo agente Cenário, e utilizadas posteriormente pelos agentes Guia e Tutor para tarefas de análise e descoberta de padrões.

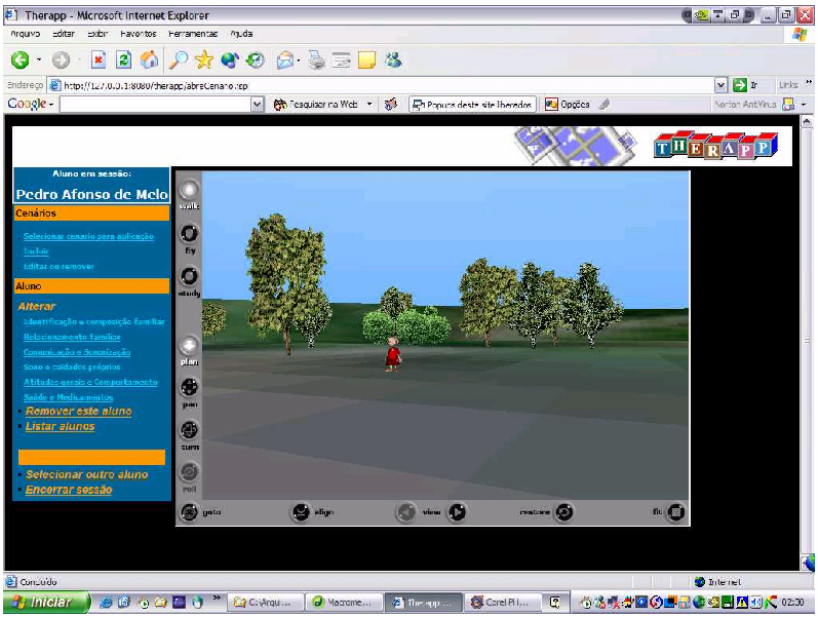


**Figura 56 -** Interface gráfica para acionada pelo agente Tutor

Diferente do sistema EC, os agentes do sistema TH trabalham de forma colaborativa, focalizando as propriedades de autonomia e colaboração do que adaptação e interação.

## 4.5
## O Protótipo MobileInvest

A aplicação MobileInvest[5] foi projetada para prover operações e informações a investidores de bolsa de valores. Através do sistema, o investidor pode selecionar e formar carteira de ações, e obter resultados *on-line* sobre o desempenho da sua carteira e das ações da bolsa de valores. O sistema captura os dados de um arquivo de cotações mantido pela BOVESPA, que disponibiliza as informações para clientes e usuários. De posse das cotações, o acionista pode comprar papéis por intermédio da corretora que ele opera. Um dos atrativos oferecidos pelo sistema MobileInvest é que o investidor pode acessar as

[5] A aplicação *MobileInvest* está sendo desenvolvida como Projeto Final de Graduação do curso de Ciência da Computação, da UniverCidade (Centro Universitário do Rio de Janeiro), unidade Ipanema, do aluno Fabiano Loubak.

informações através de dispositivos móveis, tais como celular, *palmtop*, *notebook* e outros.

O protótipo utiliza quatro agentes de software: Investidor, Corretora, Bolsa e Ação. Os agentes são autônomos e efetuam procedimentos de monitoramento, capturando periodicamente as cotações da bolsa. Eles também notificam o investidor sobre assuntos do seu interesse.

O agente Investidor é responsável pelos serviços relativos a investidores, tais como manter informações sobre o cadastro e acessar dados da corretora. O agente Investidor pode executar várias atividades, tais como comprar e vender ações de forma controlada, consultar saldo da carteira e consultar a cotação das ações. Estas atividades são executadas em colaboração com os agentes Bolsa, Ação e Corretora.

O agente Corretora é responsável pela manutenção das contas e carteiras de ações que o investidor opera. Quando a cotação de uma ação atinge um valor de venda pré-estabelecido pelo investidor, o agente Corretora notifica o Investidor, enviando uma mensagem SMP para o seu celular ou dispositivo móvel.

O agente Bolsa é responsável pela captura de informações da bolsa de valores. Ele mantém as cotações das ações sempre atualizadas, e para isto, monitora e captura as informações disponibilizadas *on-line* pela BOVESPA.

O agente Ação é responsável pela manutenção dos dados históricos de cada ação negociada. Ele mantém informações sobre a cotação, as margens de lucro por intervalo de tempo, as quantidades de compra e venda efetuadas pelo investidor ou lucro aferido nas operações. Ele também armazena o valor limite de compra e venda pré-estabelecido pelo investidor para cada ação. De posse destes valores, quando um papel atinge um valor limite determinado pelo investidor como atrativo para compra, ele notifica o agente Investidor.

Além dos agentes, o sistema MobileInvest possui alguns componentes de acesso a dados (Carteira, Estatística e Histórico) que fornecem serviços para os agentes. Dentre os serviços fornecidos, destacam-se *comprarAção*, *venderAção*, *consultarSaldo*, *consultarCotação* e *consultarDesempenho*. Os serviços Web são gerados e registrados de forma automática pelo agente Broker.

## 4.6
## Discussão

O principal propósito dos estudos de caso apresentados foi demonstrar que todas as funcionalidades e propriedades do *framework* foram exercitadas. As aplicações Expert Committee, Therapp e MobileInvest caracterizam aplicações baseadas em agentes que atuam em diferentes domínios. O protótipo Academic.Net foi considerado um sistema externo, e exercita somente a propriedade de interoperabilidade via serviços Web. O protótipo Academic.Info simula o funcionamento de um sistema de informações acadêmicas, composto por componentes reativos.

A Tabela 4 mostra os protótipos implementados e as propriedades e conceitos exercitados. As propriedades e conceitos foram reunidos em três grupos: infraestrutura, suporte ao projeto detalhado dos agentes e serviços Web/interoperabilidade. O grupo de infraestrutura focaliza os aspectos que exercitam os modelos de referência e a capacidade de evolução da arquitetura. O grupo de suporte ao projeto detalhado dos agentes focaliza os aspectos que exercitam e exercem impacto no projeto e implementação dos agentes. O grupo interoperabilidade focaliza os aspectos da arquitetura que exercitam à integração dos agentes com serviços Web e interoperabilidade com aplicações externas.

| **Propriedades Exercitadas** | **Protótipo** | | | | |
|---|---|---|---|---|---|
| | EC | AI | AN | TH | MI |
| **Infraestrutura** | | | | | |
| MOM (Message-Oriented Model) | ■ | ■ | | ■ | ■ |
| SOM (Service-Oriented Model) | ■ | ■ | | ■ | ■ |
| ROM (Resource-Oriented Model) | ■ | ■ | | ■ | ■ |
| MGT (Management Model) | ■ | ■ | | ■ | ■ |
| Configuração dinâmica | ■ | ■ | | ■ | ■ |
| Criação dinâmica de instâncias | ■ | ■ | | ■ | ■ |
| **Suporte ao Projeto Detalhado** | | | | | |
| Reuso de serviços distribuídos | ■ | ■ | | ■ | ■ |
| *Wrapper* de requisição | ■ | ■ | | ■ | ■ |
| Chamadas assíncronas | ■ | | | ■ | ■ |
| Componente *wrapper* | ■ | ■ | | ■ | ■ |
| Blackboard | ■ | | | ■ | ■ |
| Acesso a dados | ■ | ■ | | ■ | ■ |
| **Interoperabilidade Serviços Web** | | | | | |
| Plataforma → Aplicação Externa | | ■ | ■ | | ■ |
| Aplicação Externa → Plataforma | ■ | | ■ | | ■ |
| Geração automática WSDL-UDDI | ■ | ■ | | | ■ |

EC= Expert Committee AI = Academic.Info AN = Academic.Net
TH = Therapp MI = MobileInvest

**Tabela 4 -** Os protótipos implementados e as propriedades exercitadas

Exercício das Propriedades de Infraestrutura

No grupo de infraestrutura, as abstrações providas pelos modelos da arquitetura de referência foram exercitadas em todas as aplicações. O propósito principal do MOM é tornar transparente os mecanismos de comunicação. Ele envolve a participação do agente Broker e foi exercitado a partir da distribuição dos containers em diferentes *hosts* da rede. A maneira transparente e simples para efetuar uma requisição remota de serviços foi demonstrada na Seção 4.1.4.1, onde o agente Coordinator, localizado em um container AC, requisita serviços provido pelo componente ResearherData, localizado em um container CC.

As abstrações e interfaces orientadas a serviços providas pelo SOM enfatizam uma propriedade fundamental do paradigma SOA: baixo acoplamento. O SOM envolve a especificação das interfaces orientas a serviços, que são descritas em XML (ver Cap. 5, Seção 5.5.1, Fig. 71/72) garantem a propriedade de baixo acoplamento da arquitetura. As descrições são geradas de forma automática quando o usuário registra um recurso através dos assistentes GUI's.

As abstrações oferecidas pelo modelo orientado a recursos foram exercitadas na inicialização da plataforma, nas requisições de serviços e na visualização da estrutura de recursos. O agente Catalog, responsável pelas tarefas do ROM colabora com os outros agentes da arquitetura. Quando um container é iniciado, o agente C-Catalog carrega para a memória *cache* as representações descritas em XML, que são exibidas pelos assistentes GUI´s do agente Manager (Seção 4.1.5, Fig. 51). Quando ele recebe uma requisição para criar o objeto *wrapper* de requisição, o C-Manager solicita a colaboração do agente C-Catalog para recuperar o nome da entidade provedora do serviço e o endereço do container. As informações são repassadas ao agente Proxy, que cria dinamicamente a instância (Seção 3.4.1.2) e invoca a execução do serviço.

As tarefas de gerenciamento focalizam o exercício das atividades de controle do ciclo de vida, monitoramento constante da plataforma e captura de métricas e estatísticas de QoS. Estas tarefas são desempenhadas pelo agente Manager e foram demonstradas na Seção 4.1.5. Os procedimentos para o controle do ciclo de vida são assistidos por interfaces gráficas (ver Fig. 51), que facilitam o registro e visualização de recursos. As atividades de monitoramento envolvem a verificação de sincronismo e captura de métricas. Parte dos resultados destes procedimentos tais como o *status* do container, alocação de memória, quantidade

de *threads* em execução e log de transações podem ser vistos na Figura 51 deste capítulo e nas figuras 33/34, no Capítulo 3.

Exercício dos Mecanismos de Suporte ao Projeto Detalhado

Mecanismos de suporte ao projeto detalhado focalizam o exercício das atividades para facilitar o desenvolvimento de agentes. As diferentes modalidades de reuso podem contribuir de forma significativa para reduzir o número de linhas do código e diminuir o tempo de desenvolvimento. Inúmeros serviços de infraestrutura, providos pelo *framework* são reutilizados a cada nova aplicação. O objeto *wrapper* de requisição, demonstrado na Seção 4.1.4.1 pode ser citado como exemplo. Ele provê facilidades e transparência para manipular requisições de serviço, abstraindo toda a complexidade de uma requisição. Os serviços implementados pelos agentes e componentes são reutilizáveis em outras aplicações, conforme ficou demonstrado na Seção 4.2.

O objeto *wrapper* de requisição utilizado na plataforma MIDAS provê duas modalidades de chamada: síncrona ou assíncrona. Chamadas assíncronas podem trazer considerável benefício e facilitar a resolução de problemas, possibilitando que agentes possam ser programados para executar suas atividades de forma paralela. Ao efetuar uma chamada assíncrona, ele pode continuar a execução de outras tarefas sem precisar esperar. As chamadas assíncronas foram exercitadas pelo sistema EC, e demonstradas na Seção 4.1.4.1, Fig. 47.

Componentes *wrapper* foram usados intensamente pelos protótipos, e contribuíram para facilitar a implementação. Primeiro, porque eles isolam do código do agente os serviços específicos do domínio. Estes serviços se traduzem principalmente em componentes de acesso a dados, que utilizam composição de objetos da camada de dados para efetuar *queries*. A utilização de componentes *wrapper* foi demonstrada nas seções 4.1.4.2 e 4.2.

O Blackboard provê um poderoso mecanismo para auxiliar a comunicação entre agentes e as suas tarefas de *workflow*. A utilização do Blackboard foi demonstrada na aplicação EC. Durante o ciclo de vida da aplicação, os agentes consultam as vaiáveis de estado do sistema (Fig. 41) e reagem às mudanças nas variáveis adaptando o seu comportamento. O Blackboard atua também como um sensor, capturando as modificações no ambiente e notificando os agentes sobre as alterações. Estes procedimentos foram ilustrados na Seção 4.1.4.1 (Fig. 41 a 44).

As facilidades de acesso a dados providas pelo agente DBPool contribuíram para simplificar a implementação dos agentes provendo um meio simples, flexível de manipular informações em bases relacionais. Elas foram utilizadas por todos os protótipos, e demonstradas nas seções 4.1.4.1 (Fig. 49, 50) e 4.2 (Fig. 54).

Exercício da Integração Agentes-Serviços Web e Interoperabilidade

A tarefa de exercitar a propriedade de interoperabilidade bidirecional entre a plataforma e sistemas externos via serviços Web envolveu a participação de três protótipos. A interoperabilidade entre a plataforma e aplicações externas foi demonstrada no protótipo Academic.Info (Seção 4.2, Fig.52). O protótipo Academic.Net simula o papel de provedor externo. A interoperabilidade entre aplicações externas e a plataforma foi demonstrada no protótipo Academic.Net (Seção 4.3) que disponibiliza um portal na Web (Fig. 55) para recuperar informações para pesquisa de eventos passados publicadas como serviços Web pelo sistema Expert Committee. Os procedimentos para geração automática das especificações WSDL e UDDI ocorrem de forma transparente e foram exercitados nas aplicações. Eles são executados pelo agente S-Broker, na forma descrita na Seção 3.4.2.