UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação
ISSN 0103-2569

Ambiente InfoVis – Descrição da Arquitetura e Aspectos de Implementação

Milton Hirokazu Shimabukuro
Vinícius Alves Marques Branco
Maria Cristina Ferreira de Oliveira

Nº 178

RELATÓRIOS TÉCNICOS

São Carlos – SP
Nov./2002

Ambiente *InfoVis* – Descrição da Arquitetura e Aspectos de Implementação

Milton Hirokazu Shimabukuro
Vinícius Alves Marques Branco
Maria Cristina Ferreira de Oliveira

Departamento de Ciências da Computação e Estatística
Instituto de Ciências Matemáticas e de Computação
Universidade de São Paulo
Caixa Postal 668, CEP 13.560-970, São Carlos, SP

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Este documento descreve a arquitetura proposta para o *InfoVis*, um ambiente cujo objetivo é disponibilizar, de forma integrada e acessível, diversas técnicas de visualização de informação para apoiar tarefas de análise e exploração de dados. Os diferentes componentes da arquitetura são introduzidos, bem como o mecanismo de comunicação que garante a execução integrada dos diferentes componentes. Uma versão inicial da interface de acesso é apresentada, e aspectos relacionados à plataforma de implementação são discutidos.

1. INTRODUÇÃO

O projeto *InfoVis* insere-se no contexto de Visualização de Dados Exploratória e Mineração Visual de Dados, com ênfase na disponibilização de técnicas de visualização multidimensional, e no desenvolvimento de novas estratégias de visualização exploratória adequadas ao tratamento de grandes volumes de dados. A proposta consiste em (1) aplicar técnicas e estratégias de análise visual em processos de exploração e análise de grandes volumes de dados de diferentes domínios, para identificar e entender as limitações envolvidas na utilização das técnicas atuais nesse contexto; e (2) utilizar o conhecimento obtido para propor e disponibilizar, em uma plataforma aberta e acessível via *Internet*, recursos mais efetivos para apoiar tarefas de análise e exploração de dados por usuários finais, i.e., especialistas que atuam em domínios diversos.

Espera-se, com o desenvolvimento dessa plataforma, ganhar um melhor entendimento da aplicabilidade e limitações de algumas técnicas; propor extensões para tratar os aspectos de escalabilidade visual para grandes volumes de dados, incluir suporte a diferentes tipos de dados presentes nos domínios de interesse (temporais, espaciais, nominais, numéricos); e avaliar essas extensões por meio de sua aplicação na análise de conjuntos de dados representativos. O ambiente *InfoVis* deve ser acessível via *Internet*, seguindo o modelo cliente-servidor, e apresentando as seguintes características:

- Independência de plataforma. O ambiente deve ser portável entre plataformas de hardware e sistema operacional diversos;
- Extensibilidade. Deve ser possível atualizar a plataforma, com o acréscimo de novos componentes ou de versões melhoradas de componentes já existentes, com baixo impacto no restante do ambiente. Por exemplo, a adição de um novo componente deve requerer apenas uma re-compilação parcial do ambiente;
- Acessibilidade. O sistema deve ser acessível a um amplo espectro de usuários. A disponilização da plataforma na *Web* é uma maneira de ampliar a acessibilidade.
- Flexibilidade na configuração. O ambiente deve suportar a geração de configurações diversas e personalizadas de aplicativos de visualização, de acordo com as necessidades de um domínio de aplicação ou um usuário em particular;
- Integração de Múltiplas Visualizações. A possibilidade de explorar simultaneamente múltiplas visualizações de um mesmo conjunto de dados é particularmente interessante em tarefas de análise de dados exploratória. Diferentes técnicas suportam diferentes estratégias de interação, mas uma interação efetiva com múltiplas visualizações requer a coordenação das atividades de interação entre as diferentes visualizações. A estratégia denominada '*linking*' [Cle93] consiste em garantir que ações de interação executadas sobre um modelo visual sejam refletidas nos demais modelos visuais sob observação, de tal forma que uma alteração em um modelo se reflita em alterações equivalentes nos demais. É um objetivo da plataforma permitir essa integração entre as visualizações geradas pelos diferentes componentes. Para isso, cada componente da plataforma deve suportar e controlar suas formas de interação. A plataforma deve disponibilizar um mecanismo que permita aos desenvolvedores e usuários estabelecer uma ligação entre uma ação ou atividade de interação definida em um componente com alguma ação de interação correspondente definida em outro componente, possibilitando que esses componentes funcionem de forma coordenada, ou '*linked*'. Por exemplo, a ação de selecionar um item de dado em um componente pode ser ligada a uma ação de exibir detalhes sobre tal item em outro componente;
- Acesso a um amplo conjunto de fontes de dados. Para atender efetivamente um público amplo de usuários, o ambiente deve ser capaz de acessar dados de diferentes formatos ou gerenciadores de bases de dados;
- Registro de histórico. Processos de exploração visual tendem a ser fortemente interativos e iterativos, e é comum que sejam feitas várias tentativas envolvendo

diferentes parâmetros e técnicas antes que se obtenha um modelo visual interessante. Assim, um dos objetivos da plataforma é prover mecanismos para o registro dos procedimentos ocorridos ao longo do seu uso, mantendo um histórico das atividades relevantes do usuário. Esse histórico pode ser usado pelo próprio usuário para avaliar seu processo ou comportamento na exploração e análise dos dados, propiciando condições para um posterior refinamento desse processo. Outro uso para o histórico é permitir a outro usuário, local ou remoto, analisar ou replicar o processo, os resultados intermediários e a conclusão final, emitindo, então, um parecer, o que permite o estabelecimento de um cenário de trabalho colaborativo assíncrono. Futuramente, essa plataforma poderá ser estendida com o oferecimento de recursos adicionais para colaboração e para oferecer suporte à execução distribuída.

No âmbito deste projeto, a disponibilização na *Web* é explorada essencialmente como forma de ampliar a acessibilidade, mas o arcabouço proposto para a arquitetura e a opção pela implementação na plataforma Java permitem que seja explorado, futuramente, o aspecto da distribuição. Tanto a execução distribuída de componentes de visualização como o acesso a dados distribuídos oferecem opções interessantes para tratar o problema da escalabilidade em visualização, e a arquitetura proposta permite que essas opções sejam exploradas no futuro.

Nesse documento apresentamos o arcabouço conceitual da arquitetura extensível do ambiente *InfoVis*, que define como os componentes estarão organizados e se comunicarão entre si. O documento está organizado da seguinte maneira: No Capítulo 2 é apresentada uma visão geral do ambiente, do ponto de vista dos potenciais usuários de visualizações. É apresentado um protótipo inicial, ainda não funcional, da interface com o usuário que dá acesso ao ambiente. No Capítulo 3 é discutida a arquitetura de um aplicativo de visualização configurado a partir do ambiente. Os componentes da arquitetura responsáveis pelo gerenciamento dos componentes funcionais do aplicativo são apresentados. No Capítulo 4 é justificada a escolha de Java como plataforma de implementação, e os aspectos de desempenho e suas implicações são comentados. Finalmente, no Capítulo 5 são discutidos aspectos de implementação, buscando estabelecer as bases para o desenvolvimento conjunto desse ambiente.

2. O AMBIENTE *InfoVis*

Um usuário acessa a plataforma por meio de uma interface disponibilizada em um servidor de recursos de visualização, acessível na *Web*. O servidor oferece um conjunto de componentes funcionais que podem ser configurados por um usuário em um aplicativo de visualização que pode ser executado localmente na máquina cliente, independentemente da plataforma operacional. O termo componente é usado aqui não no sentido estrito atribuído em Engenharia de Software[1], mas para indicar genericamente um módulo de software (implementado como uma hierarquia de classes Java) que implementa uma certa funcionalidade. Por exemplo, um componente funcional implementa uma técnica de visualização e as ações de interação que ela suporta, enquanto que um componente básico é responsável por diferentes tipos de tarefas de gerenciamento do ambiente.

O processo de configuração de um aplicativo de visualização envolve duas tarefas: (1) a escolha dos componentes funcionais de interesse para o usuário final do aplicativo, de acordo com o domínio de aplicação; e (2) a definição de como essas visualizações podem ser 'ligadas', i.e., quais ações de interação serão refletidas simultaneamente em quais visualizações. Assim, o usuário que configura os componentes funcionais em um único

[1] Componente: objeto com funcionalidade pré-definida e acessível por meio de interfaces públicas, disponíveis para uma variedade de plataformas e ambientes de desenvolvimento. Exemplos incluem componentes *ActiveX*, *JavaBeans* e *Enterprise JavaBeans*.

aplicativo precisa ter algum conhecimento, em nível conceitual, sobre a arquitetura e como ela opera, bem como sobre os recursos de interação suportados pelas técnicas de visualização implementadas. Entretanto, uma vez configurado[2] um aplicativo segundo as necessidades de um certo grupo de usuários, sua arquitetura torna-se transparente, bastando que os usuários finais tenham conhecimento operacional das técnicas que irão utilizar. Esses usuários não precisam conhecer a arquitetura da plataforma. Pode-se, inclusive, pensar em deixar disponíveis no servidor aplicativos já finalizados, já testados e consolidados para uma determinada situação.

A Figura 2.1 apresenta uma visão de alto nível da arquitetura organizacional da plataforma *InfoVis*, que disponibiliza duas categorias de componentes: componentes do Núcleo Básico, que é comum a todos os aplicativos; e um conjunto de componentes que integram o Núcleo Funcional – Geral, no servidor, e Específico, no cliente – denominados componentes funcionais. O Núcleo Funcional Geral inclui os componentes que implementam todas as técnicas desenvolvidas e integradas ao ambiente, e o Núcleo Funcional Específico inclui as técnicas de interesse selecionadas para execução na máquina cliente de um usuário final. Os componentes funcionais podem ser componentes de manipulação de dados, que disponibilizam operações de acesso a dados externos, bem como processamento e filtragem de dados, entre outras; ou componentes de visualização, que disponibilizam técnicas de visualização e recursos de interação associados.

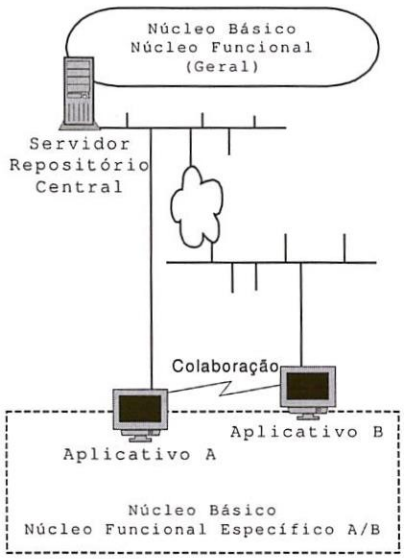

Figura 2.1 – Visão de alto nível da arquitetura organizacional da plataforma *InfoVis*.

[2] XML vem obtendo uma certa popularidade para armazenar arquivos de configuração. Todavia, Java já oferece a classe *java.util.Properties*, a qual representa uma forma tradicional, simples e eficaz de configurar programas, por exemplo, pelo estabelecimento de valores padrão para parâmetros de chamadas a métodos.

Cada componente funcional opera de forma independente dos demais – exceto pela possibilidade de definir ações de interação coordenadas com outros componentes, como discutido adiante. Dessa forma, cada componente deve disponiblizar sua própria interface com o usuário, sendo que todos devem adotar um padrão ('estilo') de interface comum. Da mesma forma, para serem integrados à arquitetura, todos devem implementar um conjunto de métodos pré-definidos que definem um padrão de interfaceamento para comunicação com os gerenciadores do Núcleo Básico, que gerenciam a execução integrada, como apresentado no Capítulo 5.

Um terceiro tipo de componente funcional são os componentes de interação, que operam associados a um componente de visualização. Assim, por exemplo, uma interação do tipo *dynamic queries* (consultas dinâmicas) [Shn94] pode ser implementada como um componente de interação independente, mas é acessível por um usuário final vinculada a uma técnica de visualização. Ao contrário dos outros tipos de componentes funcionais, um componente de interação não é visível pelo usuário enquanto componente. Tais componentes podem ser implementados como parte integrante de componentes de manipulação de dados, ou como componentes separados que interagem com outros componentes por meio de ações coordenadas.

Ambos os núcleos, Básico e Funcional, estarão disponíveis em um *site* na Internet, sendo acessíveis por uma interface que permite a usuários configurar e baixar seus aplicativos. Até o momento, a interface está sendo tratada como um aplicativo independente, sendo que a sua integração a um servidor *Web* é objeto de um projeto de mestrado recém-iniciado [Pla02]. A Figura 2.1 ilustra, na caixa pontilhada, a possibilidade de cooperação entre usuários remotos trabalhando localmente com diferentes aplicativos, por meio da troca de 'históricos' de processos de visualização. A seguir, é apresentada a proposta inicial de uma interface de acesso a esse arcabouço, i.e., aos núcleos Básico e Funcional. Uma descrição do papel desses núcleos na arquitetura de um aplicativo de visualização é apresentada no Capítulo 3.

O ambiente oferece uma interface integrada que permite a um usuário selecionar componentes funcionais entre os que estão disponíveis, e configurar esses componentes descrevendo como suas visualizações devem ser 'ligadas', i.e., quais ações de interação em um componente estarão associadas a quais ações de interação em outro. A Figura 2.2 apresenta uma primeira versão da interface. A tela é dividida em três regiões, que viabilizam o diálogo com cada um dos gerenciadores do Núcleo Básico, que são três: um Gerenciador de Componentes, um Gerenciador de Ações, e um Gerenciador de Históricos. O quarto

componente do Núcleo Básico é o Controlador da GUI Principal, responsável pela interação desse Núcleo com o usuário. O papel desses componentes, discutido mais detalhadamente no Capítulo 3, pode ser inferido a partir de uma análise da interface ilustrada nas Figuras 2.2 e 2.3.

A área superior da interface exibe o resultado do diálogo do usuário com o Gerenciador de Componentes: nessa área são listados os componentes ativos, i.e., aqueles que foram selecionados pelo usuário. A ativação de componentes ocorre por meio de um menu, ativado na opção apresentada no topo, o qual lista todos os componentes disponíveis. A Figura 2.3 ilustra a ativação de um menu (trata-se de um menu hipotético, uma vez que ainda não há componentes funcionais implementados).

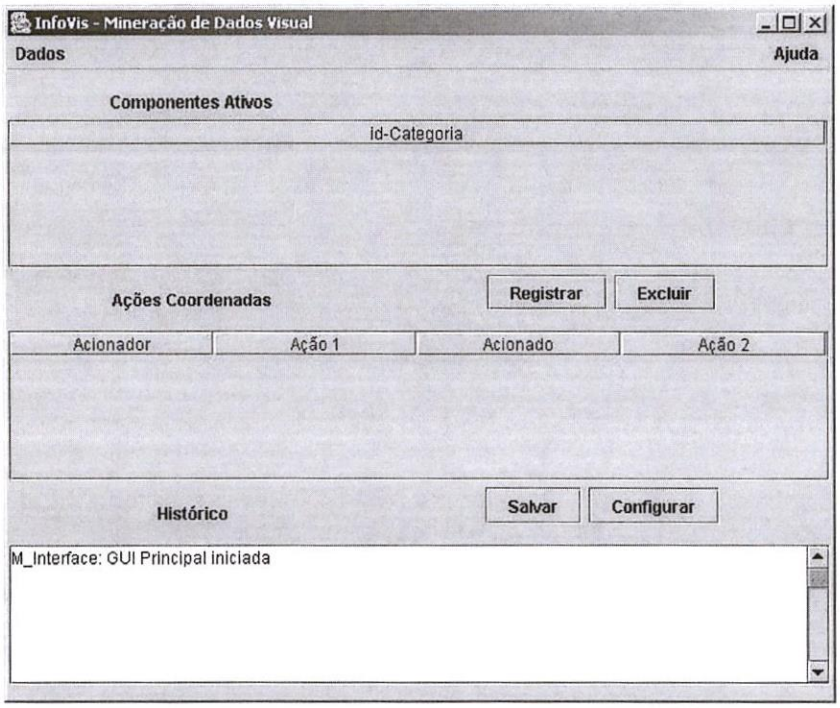

Figura 2.2 – Interface da plataforma *InfoVis*.

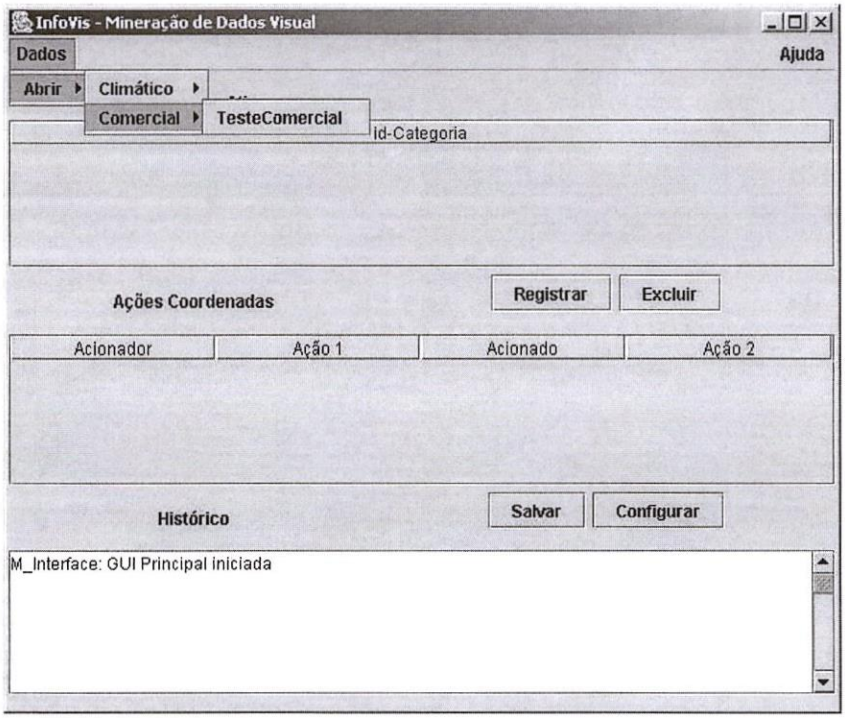

Figura 2.3 – Interface da plataforma *InfoVis* após a ativação do menu de componentes disponíveis.

Para permitir um sistema extensível foi utilizado o recurso de criação de menus do Java [Jav02], que permite que o conteúdo e hierarquia dos itens de um menu sejam estabelecidos na inicialização do aplicativo diretamente a partir de um arquivo texto de configuração. Com isso, os objetos – ou componentes funcionais – que integram o *InfoVis* são instanciados dinamicamente e não precisam ser conhecidos em tempo de compilação. A medida em que novos componentes são acrescentados à arquitetura, o arquivo de configuração dos menus é alterado de acordo e a interface atualizada na próxima inicialização, sem necessidade de re-compilação dos componentes já disponíveis. Adicionalmente, as classes podem estar distribuídas em diferentes diretórios, devidamente referenciados na variável CLASSPATH do Java. A Figura 2.4 exemplifica um arquivo de configuração de menu (à esquerda) e descreve como o mesmo é organizado (à direita) A Figura 2.5 ilustra a interface após a ativação de três componentes hipotéticos, do tipo Fonte de Dados.

0Dados 1Abrir 1Climático 2TesteClima AbrirArquivo 4 1Comercial 2TesteComercial AbrirArquivo 4 4	O menu é dividido em itens identificados pelo número que precede cada item de menu no arquivo de configuração 0 : Item constante da barra de menu principal 1 : Submenu suspenso 2 : Item ligado a um componente; a linha subseqüente identifica a classe que deve ser instanciada 3 : Item ligado a um aplicativo ou comando externo; a linha subseqüente identifica a linha de chamada para este item externo (não consta do exemplo ao lado) 4 : Final do escopo de um submenu

Figura 2.4 – Exemplo de arquivo de configuração de menu (a esquerda) e explicação da organização geral desse tipo de arquivo (a direita).

A área intermediária da janela de interface das Figuras 2.2 e 2.3 exibe o resultado do diálogo do usuário com o Gerenciador de Ações, bem como o conjunto de ações coordenadas entre os componentes ativos configurado pelo usuário. Cada componente suporta um conjunto de ações de interação (o qual é obtido ativando o menu 'ações' de cada componente no exemplo hipotético). A configuração de uma ação coordenada permite, por exemplo, que um usuário 'conecte' duas visualizações, i.e., especifique que uma determinada ação disparada em um componente funcional (que implementa uma técnica de visualização) dispare uma outra ação em outro componente funcional. Por exemplo, na Figura 2.6 a informação registrada na área visual do Gerenciador de Ações informa que uma ação de *seleção* disparada em 'componente-0' deve disparar uma ação de *brushing* no 'componente-1'. A 'ligação' poderia também ser estabelecida entre outros tipos de componentes funcionais, como Fontes de Dados e Interações Genéricas.

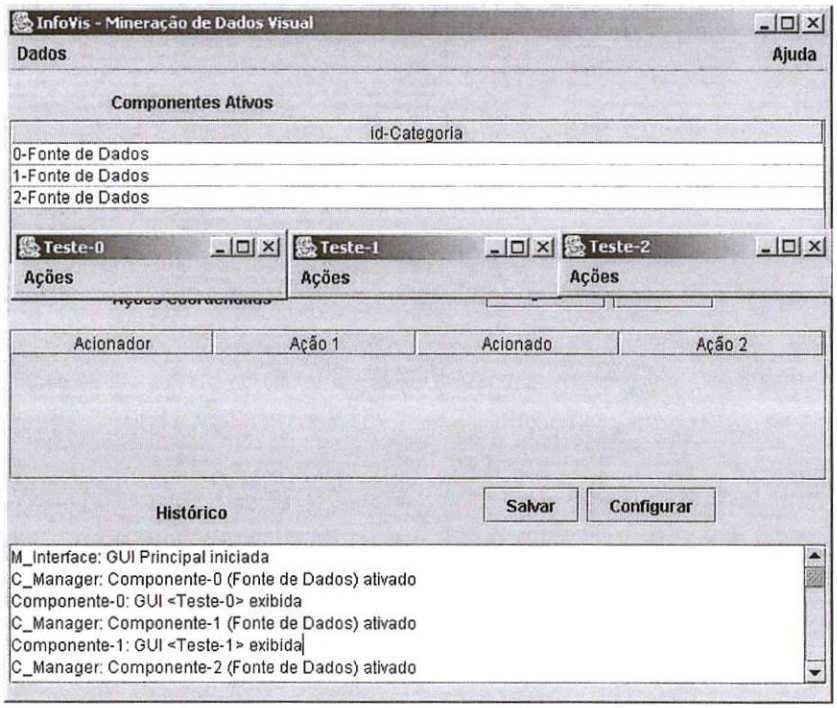

Figura 2.5 – Interface da plataforma *InfoVis* após a ativação de três instâncias de componentes funcionais disponíveis (fictícios).

Para registrar uma ação coordenada, o usuário clica no botão 'Registrar', o que vai exibir o diálogo na área inferior direita da Figura 2.6. Esse diálogo permite ao usuário selecionar o primeiro componente (o 'acionador'), em seguida selecionar, entre as ações de interação suportadas por esse componente, a ação desejada; depois selecionar o segundo componente (o 'acionado') e a ação de interação correspondente. Cabe ao usuário garantir uma semântica de aplicação correta, 'ligando' ações de interação de forma consistente. Como discutido no Capítulo 3, o Gerenciador de Ações é responsável por transferir as informações necessárias para viabilizar a ligação entre os componentes.

A área inferior da interface (Figuras 2.2 e 2.3) lista o histórico de ações realizadas: quais componentes foram ativados, quais ações coordenadas foram registradas, e em seguida que ações foram ativadas em quais componentes.

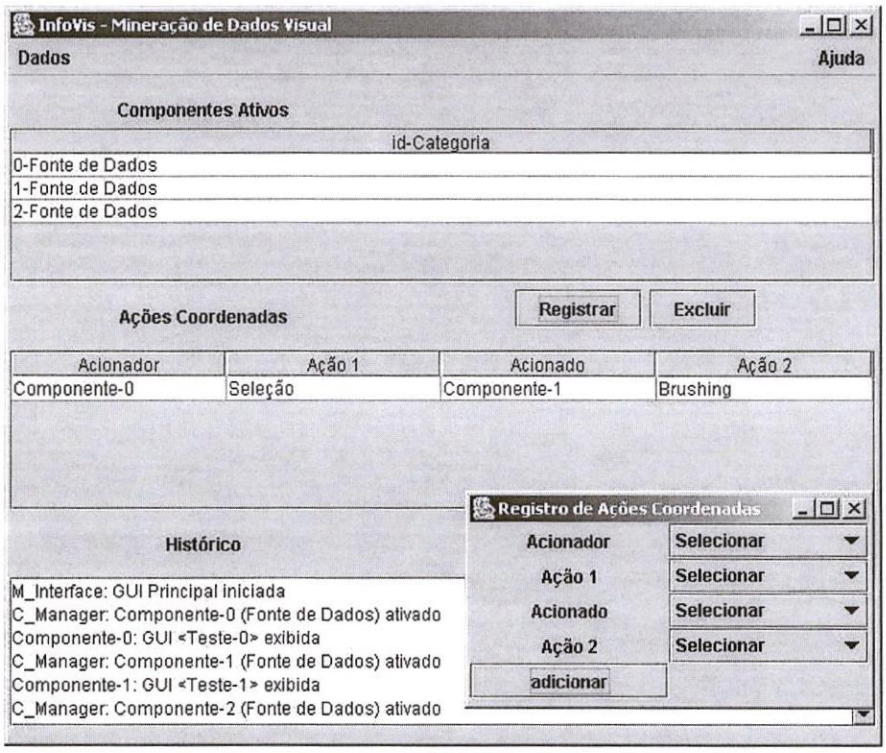

Figura 2.6 – Interface da plataforma *InfoVis* com o diálogo para registro de ações coordenadas (área inferior direita). Foi registrado que a ativação de ações de seleção em 'componente-0' deve disparar uma ação de 'brushing' em 'componente-1'.

Futuramente, essa interface poderia ser estendida para incorporar recursos mais sofisticados, por exemplo:

- Uma interface de programação visual para que o usuário configure os componentes que devem integrar seu aplicativo, por exemplo fazendo *drag and drop* e conexão visual entre componentes;
- Um editor de macros que permita ao usuário armazenar e reproduzir uma seqüência de operações ou atividades envolvidas em uma análise, evitando a repetição do procedimento passo a passo;
- Uma ferramenta para o 'empacotamento' de uma determinada análise visual registrada em um histórico, possibilitando a sua transmissão para um outro usuário, como forma de apoiar trabalho cooperativo assíncrono;
- Um analisador ou editor visual para o apoiar a interpretação de históricos;
- Um componente gerenciador de ajuda, que permita gerar material de ajuda destinado a um aplicativo específico, abrangendo apenas os componentes instalados, ou um uma ajuda genérica relativa à plataforma como um todo.

Os componentes gerenciadores básicos, juntamente com o Controlador da GUI Principal, mais um conjunto de componentes funcionais ativos e suas ações coordenadas associadas, definem um aplicativo de visualização, o qual poderá ser 'empacotado' e 'descarregado' pelo usuário a partir do servidor.

3. ARQUITETURA DOS APLICATIVOS

Como discutido no Capítulo 2, a arquitetura prevê dois tipos de componentes, agrupados em um Núcleo Básico e um Núcleo Funcional. Um usuário localizado em uma máquina cliente deve poder requisitar ao servidor o fornecimento de componentes do Núcleo Funcional para 'montar' o seu próprio aplicativo de visualização exploratória, selecionando componentes funcionais que implementam diferentes técnicas de visualização/interação de acordo com suas necessidades e recursos, ou seja, de acordo com a funcionalidade desejada do aplicativo. Os componentes funcionais implementam técnicas de visualização e interação, de forma independente, bem como técnicas específicas de acesso e processamento de dados. Os componentes funcionais podem ser agrupados em diferentes categorias, de acordo com o tipo de funcionalidade. Inicialmente, estão previstos os seguintes tipos de componentes funcionais:

- Fonte de Dados (*Data Source*), que implementam recursos para acesso a arquivos de dados ou gerenciadores de bases de dados externos, bem como recursos para filtragem, amostragem, seleção, e pré-processamento dos dados;
- Visualizador (*Visualizer*), que implementam técnicas de visualização e seus recursos de interação;

- Interação Genérica (*Data Interactor*), que implementam estratégias de interação genéricas, i.e., aplicáveis a diferentes técnicas de visualização (como o *Dynamic Queries*, por exemplo).

Já os quatro componentes do "Núcleo Básico" integram todas as configurações de aplicativo, pois são responsáveis pelas tarefas de gerenciamento dos serviços oferecidos pelos componentes funcionais e pela interação do usuário com a GUI principal. Os seguintes elementos compõem o Núcleo Básico de um aplicativo *InfoVis*: o Controlador da Interface Principal, um Gerenciador de Componentes, um Gerenciador de Ações, e um Gerenciador de Históricos, como ilustrado na Figura 3.1. Esses elementos comunicam-se entre si e possuem um canal de comunicação direta com cada componente funcional. Os componentes funcionais, por sua vez, comunicam-se indiretamente por meio do Gerenciador de Ações, o que é indicado na figura pela seta tracejada. As características de implementação e comportamento de cada Gerenciador integrante do Núcleo Básico são descritas a seguir.

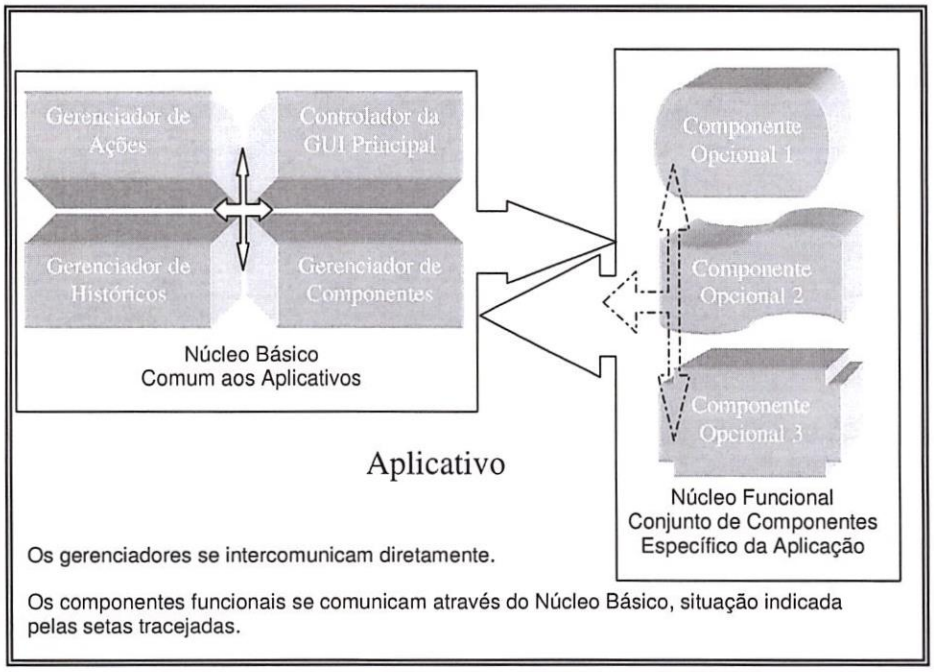

Figura 3.1 – Arquitetura genérica de um aplicativo do *InfoVis*.

Cada componente deve ser auto-suficiente, ou seja, deve manipular todas as variáveis, métodos e eventos que fazem parte do seu escopo de atuação. O componente deve informar o "Gerenciador de Componentes" sobre a sua desativação, quando tal fato ocorrer. Para integrar a arquitetura, um componente deve adotar uma estrutura padrão, por exemplo,

disponibilizando um conjunto de métodos padronizados. Uma versão inicial dessa estrutura padrão, que está sendo objeto de definição, é apresentada no Capítulo 5.

Gerenciador de Componentes

Um componente ativo corresponde a um objeto que instancia uma classe definida na arquitetura. O gerenciador de componentes é responsável por armazenar as informações sobre as classes e aplicativos externos disponíveis, instanciar objetos, controlar a desativação de objetos, e fornecer informações sobre os objetos ativos. Assim, um objeto é instanciado pelo gerenciador de componentes, que mantém armazenadas as referências a todos os objetos ativos. Cada objeto deve ter um método padrão para a sua exibição; esse método é chamado pelo gerenciador de componentes. A partir daí, a interação com o usuário passa a ser controlada pelo próprio objeto.

Gerenciador de Ações

Responsável pelo mecanismo de ligação (*linking*) entre visualizações de um mesmo conjunto de dados geradas por diferentes componentes, por meio de ações coordenadas. Em outras palavras, ações executadas sobre um modelo de visualização gerado por um componente podem ser 'propagadas' para um outro modelo de visualização dos mesmos dados gerados por outro componente. A comunicação entre os componentes relacionados é feita por este gerenciador, como ilustrado na Figura 3.2.

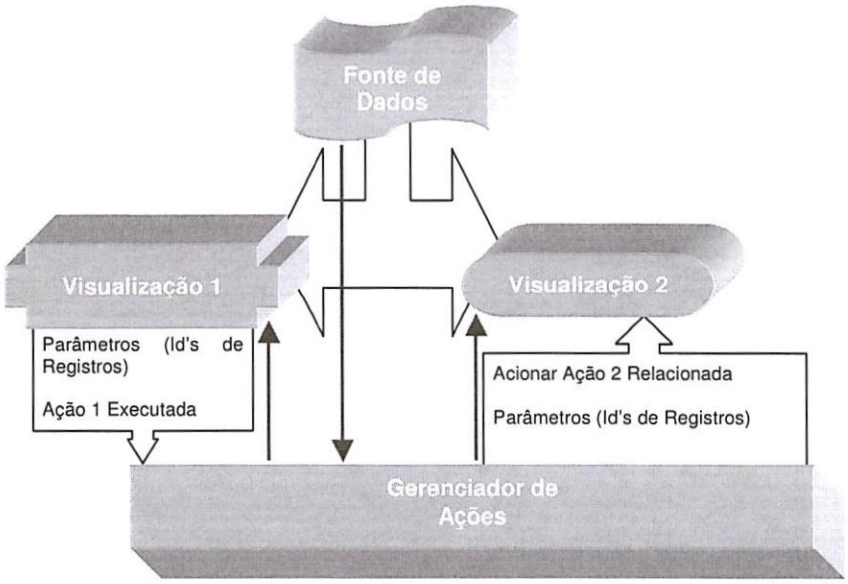

Figura 3.2 – Comunicação entre componentes no *InfoVis*, intermediada pelo Gerenciador de Ações.

A Figura 3.2 ilustra o processo de comunicação entre dois componentes de um aplicativo que disponibilizam técnicas de visualização distintas. Hipoteticamente, os componentes foram usados para gerar dois modelos de visualização distintos do mesmo conjunto de dados, sendo que ambos, denotados por 'Visualização 1' e 'Visualização 2', respectivamente, estão sendo exibidos na tela. Ao ser executada a "Ação 1" no componente responsável pela "Visualização 1", este componente informa o "Gerenciador de Ações", que por sua vez envia uma mensagem para o componente responsável pela "Visualização 2" solicitando o acionamento da "Ação 2" e repassando os "Parâmetros" com as identificações dos registros a serem processados. As setas em azul ilustram uma outra situação, na qual é a "Fonte de Dados" que dispara um conjunto de ações coordenadas sobre os dois modelos de visualização.

As ações coordenadas devem ser previamente registradas junto ao gerenciador de ações. Cada componente deve informar o conjunto de ações que é capaz de realizar e os parâmetros necessários à sua execução (por exemplo, identificadores de registros a serem processados). As ações são efetuadas sobre os registros indicados em "Parâmetros".

Gerenciador de Históricos

Responsável por fornecer meios de efetuar o registro do histórico de eventos relevantes disparados durante a interação do usuário com um aplicativo. A solicitação de registro, juntamente com o respectivo conteúdo, é feita por cada componente. Em um primeiro momento, registros podem ser mantidos em arquivos texto, mas a linguagem XML [XML] pode vir a oferecer uma opção interessante como padrão para troca de informação entre componentes.

4. PLATAFORMA DE DESENVOLVIMENTO

O arcabouço proposto será implementado na plataforma Java, que não oferece apenas uma linguagem de programação, mas um conjunto de soluções integradas para vários problemas que são críticos na arquitetura do *InfoVis*. Apesar de ser extensa e complexa, espera-se que a escolha de Java permita um bom aproveitamento do esforço despendido na concepção de uma solução flexível e genérica. Assim, os componentes dos núcleos Básico e Funcional serão implementados como classes Java. Adicionalmente, pretende-se adotar Java3D [J3D02] para implementar os recursos gráficos, JDBC [JDB02] para permitir conexão com bancos de dados externos, XML/SOAP [XML, Scr00] como protocolos para troca de informação entre diferentes componentes. Futuramente, RMI [Eck00] pode oferecer os recursos necessários à distribuição dos componentes e aplicativos. Os aspectos determinantes para a escolha da plataforma Java estão diretamente relacionados aos requisitos apresentados no Capítulo 1, que ficam naturalmente garantidas com a opção pelo desenvolvimento puramente em Java. Particularmente, a necessidade de disponibilização na *Web*, a independência de plataforma, a possibilidade de execução distribuída no futuro, o acesso transparente a diferentes fontes de dados.

Esse último aspecto é essencial devido à natureza prioritária dos requisitos de flexibilidade e portabilidade. Desta forma, o desenvolvimento do ambiente recai em um dos problemas mais críticos relacionados a bases de dados: a grande diversidade de produtos e

formatos de bancos de dados, resultado da competição comercial entre os vários fabricantes, ou das diferentes estruturas de armazenamento adotadas. Seguindo a diretiva da linguagem de programação *Java* de ser independente de plataforma, a API (*Application Program Interface*) de acesso de dados JDBC™ (*Java DataBase Connectivity*) permite que não haja preocupação, durante a fase de programação, sobre qual será a base de dados utilizada [JDB02].

Entre as vantagens e características chave de JDBC pode-se citar: acesso total a metadados (dados sobre os dados, tais como tipo, tamanho, etc); ausência de necessidade de instalações especiais (em ambientes distribuídos); compartilha a característica Java de ser fortemente orientada a objetos, encorajando a manutenção de uma hierarquia organizada de componentes; conectividade a uma ampla gama de servidores de bancos de dados; a documentação das classes padrão da linguagem Java facilita a identificação dos métodos que podem ser executados (universalidade); viabiliza a prototipação rápida, o que favorece a execução da "flexibilidade de configuração" desejada para o *InfoVis*, como discutido no Capítulo 1 deste documento. Outra característica do JDBC, comum às *APIs Java* em geral, é sua concepção voltada à simplicidade, que nesse contexto diz respeito à forma natural/lógica de execução de procedimentos. Para a obtenção de dados de uma base de dados, por exemplo, é necessário: conectar à base, criar uma instrução (*statement*), executar uma consulta e ver os resultados. Em vista das vantagens acima, fica claro que a adoção de tal tecnologia vem de encontro às características desejadas para o ambiente *InfoVis*.

Retomando o contexto geral da opção pela plataforma Java, uma desvantagem em potencial dessa plataforma é o desempenho, um aspecto que tende a ser ainda mais crítico em se tratando de um ambiente de visualização exploratória. Um código fonte Java é transformado em uma representação intermediária – *bytecodes* – que é convertida em tempo de execução, pela JVM (*Java Virtual Machine*), para o respectivo código nativo, provendo portabilidade ao código. Entretanto, quando a conversão é feita em um esquema puramente interpretado, o desempenho é baixo, e esse é um dos pontos mais criticados na plataforma Java. Entretanto, novos recursos vêm sendo incorporados à plataforma para melhorar esse aspecto e aumentar a gama de aplicações que podem ser implementadas [Arm98, Man98, Jav01]. Em particular, melhor desempenho, mantendo a portabilidade, pode atualmente ser obtido com um compilador JIT (*Just-In-Time*) ou com a máquina virtual *HotSpot*, disponível a partir da versão 1.3 do Java.

Um compilador JIT converte os *bytecodes* para código nativo, e introduz algum grau de otimização na primeira vez que sua execução é requisitada, mantendo em memória o resultado da conversão para uso posterior e evitando assim a necessidade de nova conversão.

Nesse caso, diferente do esquema de um compilador tradicional, a conversão é feita em tempo de execução, e apenas dos trechos requisitados, podendo ser até 50 vezes mais rápida do que em um esquema de interpretação puro [Arm98]. Mangione [Man98] apresenta resultados de testes comparativos entre C++ , Java (JIT) e Java (interpretação de *bytecodes*), cujos tempos de execução de algumas operações são listados na Tabela 4.1, considerando apenas o tempo de execução das instruções relacionadas.

No esquema do *HotSpot*, a conversão para código nativo ocorre de forma mista. Na primeira vez, os *bytecodes* são processados pelo interpretador, e a execução do programa é monitorada e analisada procurando identificar "*hot spots*" para aplicar, adaptativamente, um esquema de compilação com otimização [Jav01, Arm98]. Para uma melhor adaptação a diferentes características de desempenho, o compilador *HotSpot* é disponibilizado para dois perfis: cliente e servidor. O primeiro é uma atualização da máquina virtual clássica e de compiladores JIT de versões anteriores do Java SDK, e oferece um desempenho melhorado para aplicações e *applets* pela redução de tempo de inicialização e ocupação de memória. Para o perfil *server*, a máquina virtual contém um compilador adaptativo que se mostra mais adequado a aplicações em servidores. Ambos os compiladores operam sobre uma máquina virtual comum, como indicado na Figura 4.1 [Jav01].

Tabela 4.1 – Tempos, em segundos, para a execução de tarefas por códigos gerados por um compilador C++, Java com compilador JIT e Java com interpretação de *bytecodes*. Fonte: [Man98].

Descrição do teste	Tempo (segundos)		
	C++	Java (JIT)	Java (interpretação)
Repetição de **divisão com inteiro**: 10.000.000 vezes	1.8	1.8	4.8
Repetição de **divisão com ponto-flutuante**: 10.000.000 vezes	1.6	1.6	8.7
Repetição de **chamada a método estático** que contém uma divisão inteira	1.8	1.8	6.0
Repetição de **chamada a método membro** que contém uma divisão inteira	1.8	1.8	10.0

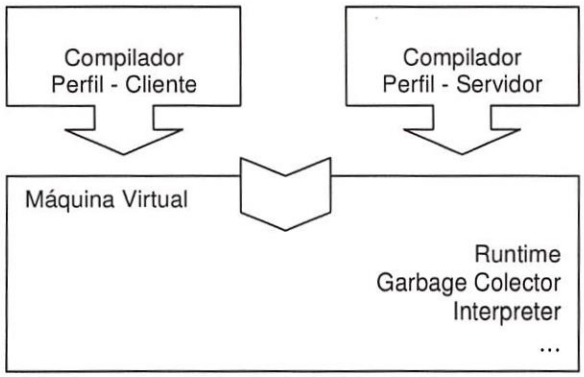

Figura 4.1 – Configuração da máquina virtual *HotSpot* que permite a escolha de um perfil para melhor adequação à aplicação. Fonte: [Jav01].

Em *java.sun.com/j2se/1.4/datasheet.1_4.html* são apresentadas informações sobre ganhos de desempenho na versão 1.4 da plataforma *Java Standard Edition*, por exemplo, no sistema de I/O e também no Java2D, permitindo que partes de aplicações sejam convertidas para código Java mantendo desempenho compatível.

Nossa visão é que a disponibilidade desses novos recursos torna menos crítica a questão do fraco desempenho de tempo de execução do Java, e que as vantagens de utilizar a plataforma superam a necessidade de tratar uma eventual perda de desempenho. Essa questão do desempenho, na verdade, vai além da discussão relativa ao código interpretado versus compilado, já que desempenho é influenciado por outros fatores, como algoritmos otimizados, configuração de hardware e sistema operacional, recursos de paralelismo e distribuição. Apesar da potencial desvantagem no processo de compilação, esses outros aspectos são tratáveis na plataforma Java.

5. ASPECTOS DE IMPLEMENTAÇÃO

O recurso de Java que permite que componentes sejam adicionados ao ambiente com uma compilação parcial é a reflexão. Este recurso provê mecanismos para a instanciação de objetos e a invocação de seus métodos em tempo de execução, dispensando o conhecimento, em tempo de compilação, de tais elementos. A verificação da existência da classe e da correta chamada aos métodos também é feita em tempo de execução [Lem01].

Como mencionado no Capítulo 2, para integrar o Núcleo Específico da arquitetura os componentes funcionais devem implementar um conjunto de métodos de interfaceamento que implementem as primitivas de comunicação entre os diferentes componentes. Esse conjunto de métodos é descrito na Tabela 5.1, e será refinado à medida que os testes avançarem.

Tabela 5.1 - Métodos a serem implementados por um componente funcional.

Método	Descrição
Construtor	No atual estágio, o construtor é implementado sem parâmetros
DoAction	Recebe mensagem do Gerenciador de Ações para realizar uma ação coordenada ligada a uma ação do componente acionador
ShowGUI	Torna o componente visível exibindo sua GUI
setManagers	Recebe as referências aos "Gerenciadores"
getComponentActions	Informa as ações de interação que o componente pode realizar
getComponentCategory	Informa a categoria a qual o componente pertence
SetIndex	Recebe o índice atribuído ao componente pelo Gerenciador de Componentes

addRegisteredAction removeRegisteredAction	Informação para controle de ação coordenada, utilizada para verificar a necessidade de informar o Gerenciador de Ações, para que este solicite uma ação ao componente acionado

Para garantir a operação integrada os diferentes componentes deverão ser capazes de se comunicar para garantir a execução de tarefas básicas de gerenciamento. A Tabela 5.2 lista os principais tipos de mensagens a serem trocadas pelos diferentes componentes da arquitetura para a execução de atividades básicas. Esses componentes incluem: o controlador da Interface (GUI) Principal (IP) que dá acesso ao ambiente; os gerenciadores do Núcleo Básico (GC, GA e GH); e os componentes funcionais responsáveis por acesso a dados e visualização. Está em andamento a implementação dos componentes do Núcleo Básico e dos métodos que implementam as ações de gerenciamento pelos quais esses módulos são responsáveis, bem como de um componente de acesso a dados.

Tabela 5.2 – Mensagens a serem trocadas entre os componentes da arquitetura.

Siglas IP : Controlador da GUI Principal GC : Gerenciador de Componentes GA : Gerenciador de Ações GH : Gerenciador de Históricos CP : Componente		
Atividade: Inicialização da GUI Principal		
Emissor	Receptor(es)	Tipo da Mensagem
IP	GC	Informação sobre o número máximo de componentes ativos simultaneamente [Construtor do GC]
IP	Gerenciadores	Cada gerenciador recebe as referências dos outros gerenciadores [set[Components | Actions | Log]Manager]
IP	Gerenciadores	Solicitação das áreas de visualização de informação de cada gerenciador [getActiveComponentsInfoViewer] [getCoordinatedActionsInfoViewer] [getLogBoard]
IP	GC	Informações sobre os itens do menu da GUI principal [setMenuItemsInfo]
Atividade: Ativação de um Componente		
Emissor	Receptor(es)	Tipo da Mensagem
IP	GC	Solicitação de ativação de um componente definido pela escolha no menu da GUI principal [activateComponent]
IP	GC	Solicitação de informação sobre o tipo de item de menu (aplicativo externo ou componente) [getMenuItemType]
GC	CP	Informação sobre o índice que foi atribuído ao componente [setIndex] (chamada dinâmica, em tempo de execução)
CP	GC	Solicitação para o acréscimo de descrição sobre o componente no quadro de visualização do GC [appendDescription]
GC	CP	Solicitação da categoria a qual o componente pertence [getComponentCategory] (chamada dinâmica, em tempo de execução)
IP	CP	Referências dos gerenciadores [setManagers] (chamada dinâmica, em tempo de execução)
IP	CP	Solicitação para exibição da GUI do componente [showGUI] (chamada dinâmica, em tempo de execução)
Atividade: Desativação de um Componente		
Emissor	Receptor(es)	Tipo da Mensagem

CP	GC	Aviso sobre a desativação do componente [deactivateComponent]
GC	GA	Aviso sobre a desativação do componente [setDeactivatedComponent]
GA	GC	Solicitação da referência do componente [getComponentRef]
GA	CP	Solicitação para atualização do controle de ação coordenada com o componente desativado [removeRegisteredAction] (chamada dinâmica, em tempo de execução)
Atividade: Registro de uma Ação Coordenada		
Emissor	Receptor(es)	Tipo da Mensagem
IP	GA	Solicitação para ativação do processo de registro de ação coordenada [registerCoordinatedActions]
GA	GC	Solicitação da lista de estado (ativo ou não) de componentes [getComponentStatusList]
GA	GC	Solicitação da referência do componente [getComponentRef]
GA	CP	Solicitação das ações disponibilizadas pelo componente [getComponentActions]
GA	CP	Solicitação ao componente acionador para atualização do controle de ação coordenada [addRegisteredAction] (chamada dinâmica, em tempo de execução)
Atividade: Exclusão de Ação Coordenada		
Emissor	Receptor(es)	Tipo da Mensagem
IP	GA	Solicitação para remoção de registros de ação coordenada [removeCoordinatedActions]
GA	GC	Solicitação de referência do componente acionador [getComponentRef]
GA	CP	Solicitação para atualização do controle de ação coordenada [removeRegisteredAction] (chamada dinâmica, em tempo de execução)
Atividade: Coordenação de Ação		
Emissor	Receptor(es)	Tipo da Mensagem
CP	GA	Aviso sobre a realização de uma ação coordenada [setActionDone]
GA	GC	Solicitação de referência do componente acionado [getComponentRef]
GA	CP	Solicitação o acionamento da ação coordenada [doAction] (chamada dinâmica, em tempo de execução)
Atividade: Registro de Histórico (ocorre em todas atividades)		
Emissor	Receptor(es)	Tipo da Mensagem
IP	CH	Solicitação de registro de histórico oriundo da Interface Principal [writeLogFromMI]
GC	CH	Solicitação de registro de histórico oriundo do Gerenciador de Componentes [writeLogFromCM]
GA	CH	Solicitação de registro de histórico oriundo do Gerenciador de Ações [writeLogFromAM]
CP	CH	Solicitação de registro de histórico oriundo de um componente [writeLogRecord]

REFERÊNCIAS

[Arm98] Armstrong, E.; HotSpot: A new breed of virtual machine www.javaworld.com/jw-03-1998/jw-03-hotspot.html

[Cle93] Cleveland, W.S.; *Visualizing Data*, Hobart Press, Summit, NJ, 1993.

[Eck00] Eckel, B.; Thinking in Java, 2a. edição, Prentice Hall, 2000.

[Jav01] The Java HotSpot Virtual Machine, Technical White Paper, Sun Microsystems, May 2001 java.sun.com/products/hotspot

[Jav02] The Java Tutorial. A practical guide for programmers java.sun.com/docs/books/tutorial/index.html, última atualização: 04/03/2002, Sun Microsystems.

[J3D02] Java 3D API, java.sun.com/products/java-media/3D/, última atualização: 29/07/2002, Sun Microsystems.

[JDB02] *JDBC Data Acess API*. http://java.sun.com/products/jdbc/, última atualização: 2002, Sun Microsystems.

[Lem01] Lemay, L.; Cadenhead, R. Aprenda em 21 dias: Java 2 Professional Reference, Editora Campus, 2001.

[Man98] Mangione, C.; Performance tests show Java as fast as C++ www.javaworld.com/javaworld/jw-02-1998/jw-02-jperf_p.html

[Pla02] R.G. Platz; M.C.F. de Oliveira; Concepção e Implementação do Núcleo de uma Plataforma para Visualização Exploratória, Projeto de Pesquisa de Mestrado, Proc. FAPESP 01/12530-7, 2001.

[Scr00] K. Scribner, M. C. Stiver – Understanding SOAP – The Authoritative Solution, Sams Publishing, 2000.

[Shn94] B. Shneiderman, B. Dynamic Queries for Visual Information Seeking. *IEEE Software*, Vol. 11(6), pp. 70-77, 1994.

[XML] W3Schools – XML Tutorial, http://www.w3schools.com/xml/default.asp.