2
Revisão da Literatura

Neste capítulo é apresentada uma revisão de trabalhos relacionados. Na Seção 2.1, são apresentadas abordagens que estendem tecnologias desenvolvidas para a Engenharia de Software de uma forma específica para o desenvolvimento de groupware. Uma ênfase maior é dada às abordagens voltadas para o desenvolvimento de groupware baseado em componentes, dada a proximidade à proposta desta tese.

Neste capítulo também é feita uma revisão sobre a tecnologia de componentes de software (Seção 2.2) e de engenharia de domínio (Seção 2.4), que são subsídios para os demais capítulos da tese. Na Seção 2.2, são apresentados alguns termos e conceitos do desenvolvimento baseado em componentes, resumindo o Apêndice A, que detalha a componentização de software e o uso de frameworks. Optou-se por fazer uma revisão mais extensa na forma de apêndice, dada a variedade de definições e conceitos presentes na literatura da área.

Por fim, na Seção 2.3, a abordagem proposta nesta tese é comparada com algumas plataformas de desenvolvimento de groupware componentizado encontradas na literatura.

2.1.
Abordagens para o Desenvolvimento de Groupware

Vários trabalhos na literatura estendem as abordagens, técnicas, tecnologias e ferramentas da Engenharia de Software para o desenvolvimento de groupware, de modo a torná-las mais propícias para as características deste tipo de aplicação. Sendo a abordagem proposta nesta tese voltada para o desenvolvimento de groupware e estando esta tese no contexto de um projeto de pesquisa que visa instrumentar todo o ciclo de desenvolvimento de groupware, são apresentados nesta seção alguns exemplos de abordagens. São apresentados requisitos, UML estendida, padrões de projetos, arquiteturas de groupware, frameworks,

componentes, toolkits e técnicas de avaliação heurística específicas para o desenvolvimento de groupware.

2.1.1. Requisitos de Groupware

Groupware em geral apresenta requisitos recorrentes. Diversos trabalhos na literatura [Tietze, 2001; Schmidt & Rodden, 1996; Mandviwalla & Olfman, 1994] catalogam requisitos para instrumentar a especificação, análise e avaliação de groupware. Apesar de não ser possível especificar completamente um groupware a partir de uma lista de requisitos, as listas auxiliam a iniciar o processo, instrumentando o desenvolvedor. Nesta seção, são apresentados os requisitos propostos por Tietze [2001], por serem voltados para groupware baseado em componentes. Estes requisitos são utilizados no Capítulo 5 para analisar a arquitetura proposta.

Tietze [2001] divide seu catálogo de requisitos em dois grupos: *requisitos de usuário* (RU) e *requisitos de desenvolvedor* (RD). Os requisitos de usuário impactam diretamente os usuários finais do sistema colaborativo. Os requisitos de desenvolvedor impactam os desenvolvedores que utilizam a arquitetura de componentes para criar e adaptar as ferramentas colaborativas. Tietze propõe ao todo onze requisitos de usuário e nove de desenvolvedor. Tietze apresenta os requisitos através de cenários envolvendo personagens fictícios, que são sucintamente reproduzidos a seguir.

RU1 – Acesso aos objetos compartilhados e às ferramentas de colaboração – Andrew ao chegar para trabalhar em sua estação de trabalho deseja que o ambiente ofereça acesso aos documentos compartilhados e às ferramentas colaborativas para manipular os objetos e interagir com os demais participantes, oferecendo persistência das alterações promovidas nos objetos.

RU2 – Auxílio na escolha das ferramentas apropriadas – Andrew necessita escrever um relatório técnico cooperativamente com Bárbara, localizada remotamente. Dada a variedade de tarefas e ferramentas, o ambiente auxilia na escolha da ferramenta apropriada para a realização das tarefas e para a edição dos objetos compartilhados.

RU3 – Fornecimento de informações de percepção – Ao entrar no ambiente, Bárbara visualiza que Andrew está conectado e que um determinado capítulo do relatório está sendo editado por ele. Andrew também é notificado da disponibilidade de Bárbara.

RU4 – Colaboração síncrona e assíncrona – Andrew necessita discutir uma questão com Bárbara. O ambiente fornece serviços de colaboração síncrona e assíncrona, de modo que os participantes selecionam o modo de interação mais adequado a cada situação.

RU5 – Acesso ao ambiente independente da estação de trabalho – Bárbara necessita de informações presentes no laboratório da empresa. Ela deixa sua estação de trabalho e se conecta do computador do laboratório e continua a colaboração com Andrew do ponto onde foi interrompida.

RU6 – Fornecimento de espaço privativo e público e transição entre eles – Bárbara quer rascunhar algumas observações sem que Andrew tenha acesso. Quando estiverem mais consolidadas, ela convidará Andrew para visualizar e trabalhar no texto.

RU7 – Extensão dinâmica do ambiente – Andrew e Bárbara necessitam construir um determinado tipo de figura. Andrew busca na Internet uma ferramenta para este propósito e a incorpora ao ambiente, de modo que Bárbara também tenha acesso.

RU8 – Sincronização entre ferramentas diferentes – Bárbara abre uma ferramenta para manipular a estrutura do documento, enquanto Andrew utiliza o editor de texto. As alterações feitas em uma ferramenta são refletidas na outra.

RU9 – Mobilidade – Charles, o gerente do projeto, ao sair de uma reunião com um cliente, acessa o sistema através de seu PDA para verificar o andamento e as pendências.

RU10 – Agrupamento de ferramentas – Ao encerrar a sessão de colaboração, Andrew e Bárbara agrupam as ferramentas para possibilitar restaurar o ambiente mais rapidamente na próxima sessão.

RU11 – Alta performance – Ao colaborar, os participantes esperam que seu trabalho ocorra sem atrasos devido à latência do sistema De modo geral, as

ferramentas de colaboração síncronas têm requisitos mais fortes de *feedthrough* (modificações devido a ações dos companheiros) do que as ferramentas assíncronas, e operações interativas têm requisitos mais fortes de *feedback* (modificações devido a ações do indivíduo) do que operações em lote.

RD1 – Reuso da experiência e conhecimento anteriores – David, novo membro da equipe de desenvolvimento de ferramentas, deseja que sua experiência e conhecimento sobre programação de aplicações mono-usuário sejam aproveitados.

RD2 – Aproveitamento do modelo de dados – Visando a interoperabilidade e o reuso, David aproveita modelos de dados já existentes.

RD3 – Compartilhamento transparente de dados – David não deseja preocupar-se com a distribuição e com o compartilhamento de dados pelo sistema. A infra-estrutura provê recursos para gerenciar o acesso, a alocação e o compartilhamento dos dados.

RD4 – Suporte a dados locais e compartilhados – Nem todos os dados devem ser compartilhados. David decide se o dado vai ser local ou compartilhado, e a infra-estrutura provê recursos para tratar ambos da mesma maneira.

RD5 – Acesso às informações de percepção – David necessita acesso às informações necessárias para prover aos participantes informações de percepção.

RD6 – Disponibilização de novas ferramentas – A infra-estrutura não deve oferecer dificuldades para a disponibilização de novas ferramentas.

RD7 – Escalabilidade – A performance da infra-estrutura não se degrada perceptivelmente quando novos usuários se conectam.

RD8 – Integração com ferramentas externas – A infra-estrutura é adaptável para possibilitar a integração do ambiente a ferramentas externas.

RD9 – Suporte a ferramentas localizadas no servidor – A infra-estrutura provê recursos para que ferramentas sejam executadas no servidor, como nos casos em que é necessário acesso direto a determinados recursos ou execução ininterrupta.

2.1.2. UML Estendida

Ao projetar e desenvolver software, é necessária uma notação comum entre os envolvidos para que eles possam documentar e debater sobre o projeto do sistema. A notação de projeto que se tornou padrão e largamente utilizada pela indústria é a UML (*Unified Modeling Language*). A UML é uma notação extensível que inclui definições de diagramas de modelagem para as diversas atividades do desenvolvimento, desde as primeiras até as mais refinadas [Booch et al., 2000]. Para propósitos de extensibilidade, a UML prevê o uso de estereótipos, que são pontos de extensão dos diagramas para serem utilizados com particularidades do domínio em questão. Estereótipos são representados nos diagramas através de um nome precedido por "<<" e sucedido por ">>".

Alguns trabalhos estendem a linguagem UML para a modelagem de aspectos específicos de groupware baseado em componentes. Estas extensões tornam mais direta a integração do projeto com a arquitetura utilizada. Na proposta de Rubart & Dawabi [2002], para representar aspectos específicos da arquitetura, as classes que representam objetos compartilhados são marcadas com o estereótipo <<shared>> e as que representam componentes de groupware, com o estereótipo <<component>>, conforme ilustrado na Figura 2.1. Estas extensões indicam implicitamente os mecanismos de encaixe na arquitetura, de concorrência, de sincronização, etc. Os componentes de groupware implementam as ferramentas colaborativas.

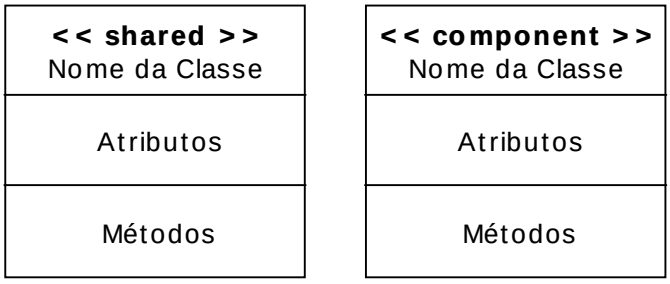

Figura 2.1. UML estendida apresentando classes que representam objetos compartilhados (à esquerda) e componentes de groupware (à direita) [Rubart & Dawabi, 2002]

A associação entre os componentes e os objetos compartilhados é realizada através de tarefas, o que leva à necessidade de representá-las no diagrama de classes. Uma tarefa é indicada através do estereótipo <<task>> nas relações entre as classes. Na Figura 2.2, dois componentes (um editor de documento e um

visualizador de estrutura) atuam em um mesmo objeto compartilhado (documento). As associações são realizadas através das tarefas editar e navegar. Na figura, um objeto da classe Documento contém outros objetos da mesma classe (um documento composto de seções). Como o objeto Documento é um objeto compartilhado, é representada a extensão <<slot>> antes dos nomes dos atributos. Este estereótipo faz a distinção entre os atributos regulares e os objetos compartilhados, que têm necessidades específicas de sincronização e integridade [Rubart & Dawabi, 2002].

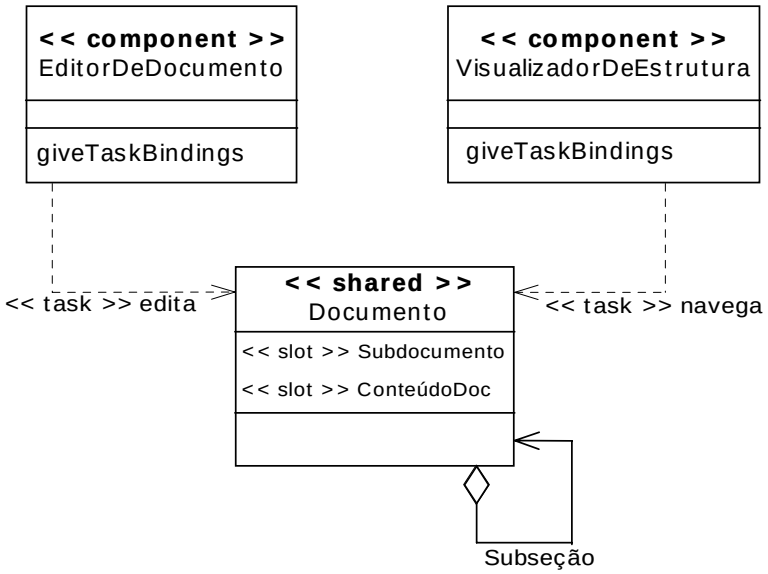

Figura 2.2. Diagrama de classes apresentando componentes de groupware que executam tarefas em um mesmo objeto compartilhado

As operações nos objetos compartilhados são realizadas durante uma sessão. Uma sessão é individual ou coletiva, sendo que a primeira pode ser transformada na segunda. Na Figura 2.3, são apresentadas duas sessões, que são representadas através do símbolo de componentes da linguagem UML marcados com a extensão <<session>>.

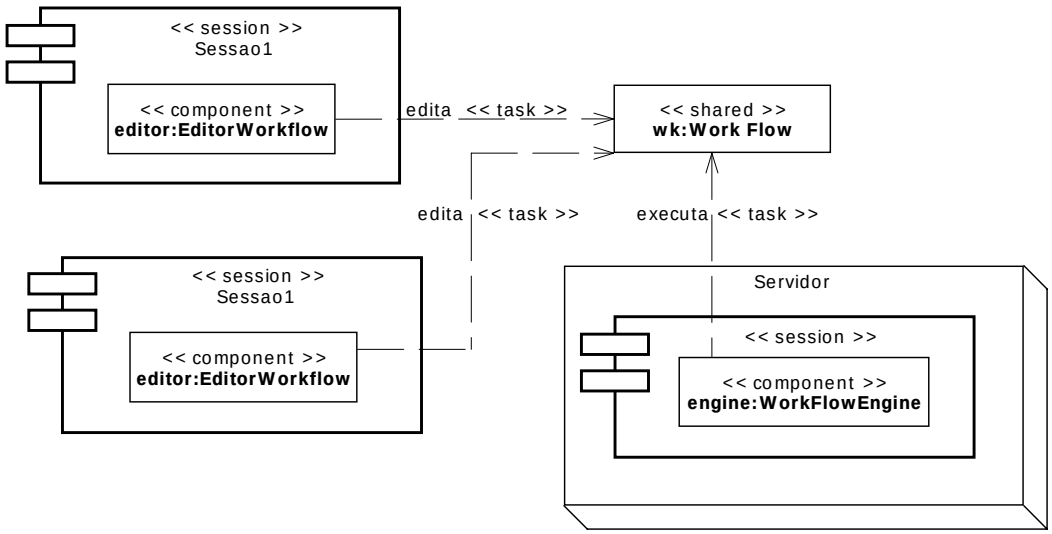

Figura 2.3. Extensão da UML para modelar sessões e nós de processamento

Para representar as operações realizadas no servidor, como por exemplo, a execução de uma instância de um fluxo de trabalho por uma máquina de workflow, utiliza-se uma sessão localizada em um nó representando o servidor, conforme pode ser observado na Figura 2.3. Quando o nó não for representado, assume-se uma estação cliente.

2.1.3. Padrões Específicos

Um padrão descreve um problema que foi identificado, definido, resolvido e documentado, direcionando a comunicação entre os desenvolvedores, o entendimento do sistema e o reconhecimento de problemas recorrentes. Os padrões catalogam problemas que os desenvolvedores enfrentaram, bem como as soluções dadas [Paludo & Burnett, 2005]. Com os padrões, a experiência dos desenvolvedores de software é documentada e reusada, registrando soluções, idéias e conceitos para um determinado problema ou contexto particular [Gamma et al., 1994]. Padrões de projeto registram uma solução recorrente ligada ao projeto do sistema, constituem uma linguagem comum entre os desenvolvedores e normalmente possuem suporte computacional implementado. Padrões de análise possuem objetivos similares aos padrões de projeto, porém com ênfase para as fases iniciais do ciclo de desenvolvimento de software. Um padrão de análise representa uma idéia que já foi útil em um contexto particular e que provavelmente será útil em outros [Fowler, 1996]. Os padrões de análise são

conceituais, com o objetivo de representar a forma de pensar do usuário em detrimento da forma utilizada pelo computador.

Alguns trabalhos propõem padrões de projetos específicos para instrumentar o desenvolvimento de groupware. Na Figura 2.4 são apresentadas famílias de padrões específicos para groupware, disponíveis em [Groupware Patterns Swiki, 2005]. Para cada padrão, é descrito intenção, família, nomes equivalentes, problema que endereça, cenário de uso, contexto, indicações, solução, participantes, racional da solução, usos conhecidos, padrões relacionados, referências bibliográficas e maneira de citar.

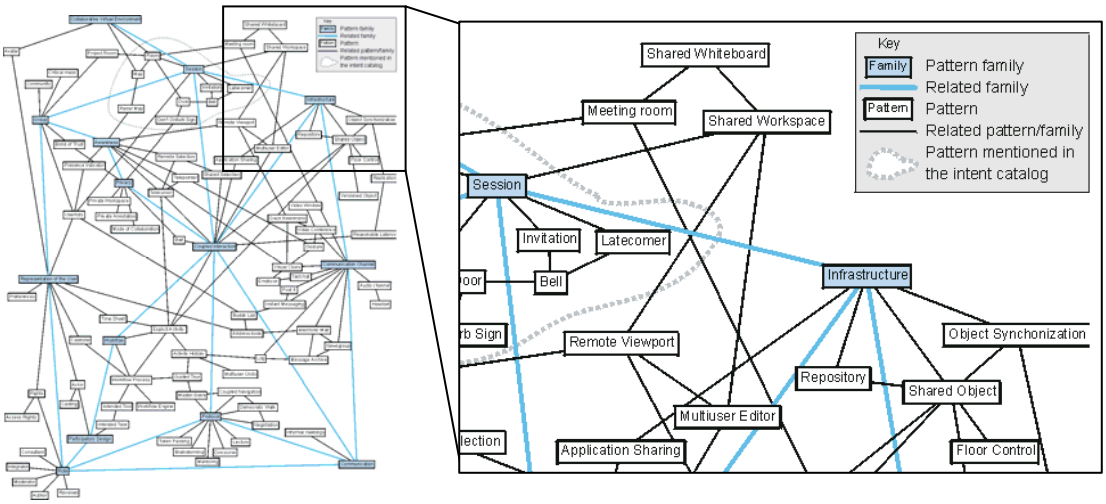

Figura 2.4. Padrões de projeto para groupware

Lukosch & Schümmer [2004] propõem uma linguagem de padrões para groupware, visando instrumentar o desenvolvimento, de modo a favorecer o reuso de soluções e capacitar mais rapidamente membros inexperientes no desenvolvimento de groupware. Também é considerada a integração com frameworks existentes, de modo a valorizar o reuso de decisões de projeto bem sucedidas e de conhecimento provido por especialistas.

Santoro et al. [2001] propõem um modelo baseado em padrões conceituais que descrevem situações e problemas comuns em um ambiente de ensino-aprendizagem e apresentam soluções recorrentes e reusáveis. Os padrões objetivam instrumentar o desenvolvedor de um ambiente no levantamento de necessidades, de modo a não deixar de fora questões importantes. Alguns dos padrões são transformados em padrões de projeto e outros servem de guia para o desenvolvimento. O modelo é apresentado em mais detalhes na Seção 3.8 do capítulo seguinte.

Kolfschoten et al. [2004] propõem uma engenharia de colaboração baseada em thinkLets, que são padrões de intervenção e concepção da dinâmica de colaboração do grupo visando incentivar um determinado padrão de colaboração. Os autores classificam os thinkLets de acordo com seu propósito principal para a fase da colaboração.

2.1.4. Arquiteturas de Groupware

Alguns trabalhos propõem arquiteturas específicas para groupware. Normalmente a arquitetura de um groupware provê mecanismos de controle de acesso, de compartilhamento, de concorrência e de sincronização entre objetos. Na Figura 2.5 encontra-se a arquitetura para groupware baseado em componentes proposta por Tietze [2001]. Cada participante utiliza uma aplicação cliente que contém os componentes de groupware. Estes componentes são instanciados a partir do repositório do *Component Broker*, localizado no servidor. Cada componente combina uma camada de interface, onde são implementadas as funcionalidades relativas à interação com o usuário, e uma camada de lógica de aplicação, onde se localizam as funcionalidades específicas do componente e da manipulação dos objetos. Os componentes acessam os objetos através do Modelo de Dados do Domínio, que implementa as funcionalidades relativas ao domínio em questão e é responsável por manter as consistências semânticas entre os objetos. O Modelo de Dados copia os objetos compartilhados localizados no Gerenciador de Objetos do servidor, que é notificado das alterações realizadas e se encarrega de replicá-las aos demais componentes. Diversos componentes, inclusive de natureza diferente, compartilham os mesmos objetos de cooperação. A arquitetura provê recursos para a propagação de alterações e o controle de concorrência, livrando os desenvolvedores destas tarefas.

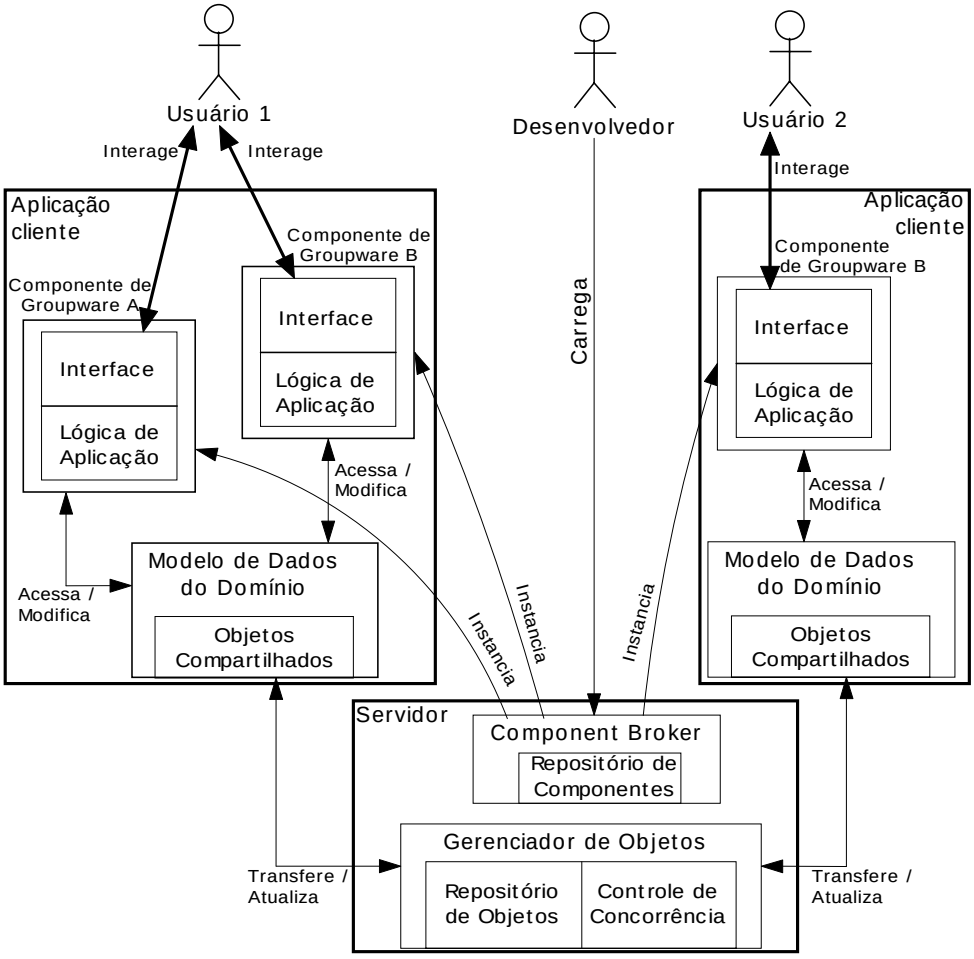

Figura 2.5. Arquitetura proposta por Tietze [2001]

A arquitetura também provê recursos para que os desenvolvedores insiram ou atualizem componentes no sistema, sem reiniciá-lo. Esta carga é feita no *Component Broker*, que é a partir de onde os clientes instanciam os componentes. O *Component Broker* encarrega-se de manter persistentes e consistentes as versões utilizadas pelos clientes, além de gerenciar as instâncias que executam no servidor. A utilização de uma arquitetura que gerencia o compartilhamento de objetos e o armazenamento e instanciação de componentes instrumentam o desenvolvedor, que por sua vez se concentra no desenvolvimento do componente como se fosse uma aplicação mono-usuário utilizando objetos locais [Tietze, 2001].

A divisão da implementação do componente em interface e lógica da aplicação possibilita que ele tenha diferentes versões da camada de interface para, por exemplo, computadores pessoais, computadores de mão e celulares. Estas implementações da camada de interface utilizam a mesma camada de lógica de aplicação, e com isto, o comportamento do componente é mantido para as

diferentes formas de acesso. Alterações no comportamento do componente se refletem nas diferentes formas de acesso.

Dewan [1998] propõe uma arquitetura genérica para sistemas colaborativos. De acordo com esta arquitetura, um groupware é estruturado em várias camadas com diferentes níveis de abstração, conforme ilustrado na Figura 2.6. A camada mais alta está relacionada ao nível semântico, específico ao domínio em questão. A camada inferior corresponde ao nível do hardware. Conforme ilustrado na figura, algumas camadas são compartilhadas e outras são replicadas nas ramificações correspondentes aos usuários conectados. As camadas comunicam-se entre si através de eventos de *feedback*, que transitam na mesma ramificação, e eventos de *feedthrough*, que são transmitidos à camada correspondente em outra ramificação.

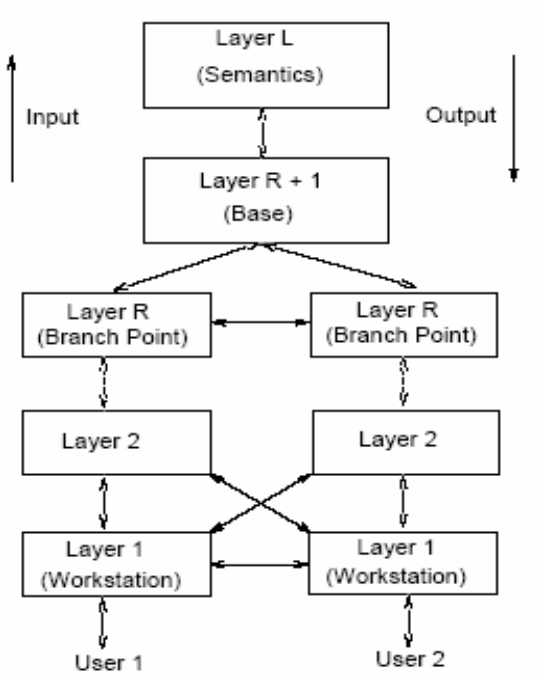

Figura 2.6. Arquitetura genérica de groupware proposta por Dewan [1998]

Laurillau & Nigay [2002] propõem uma arquitetura genérica para groupware, baseada na arquitetura de Dewan e no estilo arquitetural PAC*. O PAC* [Calvary et al., 1997] é uma extensão do PAC (*Presentation, Abstraction and Control*), que organiza a arquitetura em apresentação, que contém o código responsável pela montagem da interface com o usuário; abstração, que contém o núcleo funcional; e controle, que registra e gerencia as interdependências. No PAC*, além destas divisões, a arquitetura é separada de acordo com as classes de

funcionalidade do modelo Clover: produção, coordenação e comunicação. A Figura 2.7 apresenta o Clover Architecture.

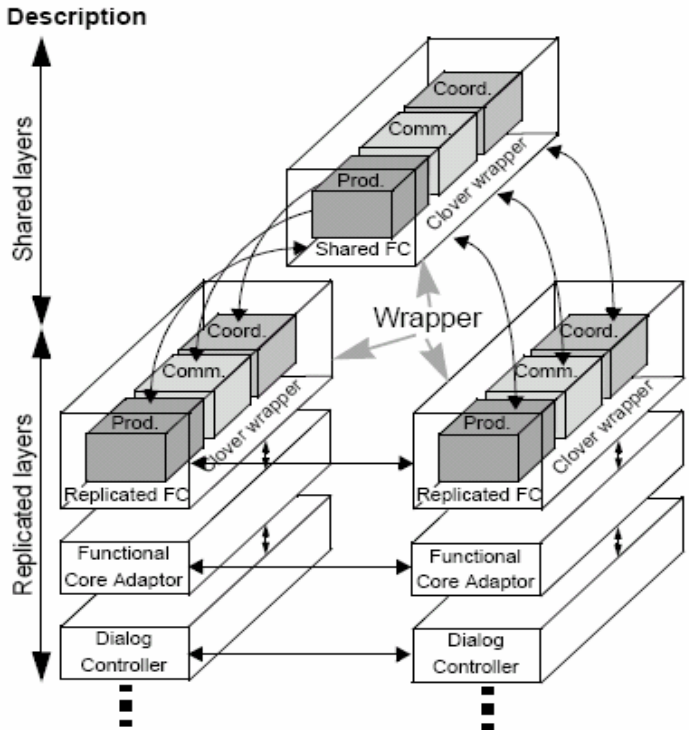

Figura 2.7. Clover Architecure [Laurillau & Nigay, 2002]

O PAC* e o Clover Architecture, assim como a abordagem proposta nesta tese, organizam a arquitetura em três classes de funcionalidades derivadas de um modelo de colaboração. Entretanto, o PAC* e o Clover Architecture oferecem um estilo arquitetural e uma arquitetura genérica para a construção de groupware, não pressupondo componentes de software compatíveis com um modelo de componentes. Nesta tese, os componentes são associados a *component frameworks* específicos. Além disto, nesta tese, tanto o groupware quanto os serviços são componentizados em função do modelo 3C, e o modelo é utilizado na organização de kits de componentes, que são construídos a partir de uma engenharia de domínio.

2.1.5. Frameworks

Um framework agiliza o processo de desenvolvimento de software que lida com determinado tipo de problema, provendo recursos que fornecem flexibilidade para os desenvolvedores adequarem-no às suas necessidades, reusando a solução

da parte comum dos problemas. De acordo com Johnson [1997], um framework é definido como um projeto reusável de todo ou de parte de um sistema, fornecendo um conjunto de classes abstratas e a forma com que suas sub-classes interagem. Um framework é um esqueleto de um sistema, que é instanciado e especializado para gerar uma família de aplicações [Govoni, 1999].

Na literatura são encontrados diversos frameworks voltados para o desenvolvimento de groupware. Prakash & Knister [1994] propõem um framework para gerência de operações em uma ferramenta colaborativa, possibilitando registrar e desfazer operações. Lee et al. [2002] propõem um framework voltado para o desenvolvimento de CVEs (*Collaborative Virtual Environments*). Kirsch-Pinheiro et al. [2002] propõem um framework para gerenciar as informações de percepção de um groupware. Nunamaker et al. [2001] propõem um framework para a gestão do conhecimento em ambientes colaborativos. Buzko et al. [2000] propõem um framework voltado para utilização da computação móvel em ambientes colaborativos. Osuna & Dimitriadis [1999] propõem um framework para o desenvolvimento de ambientes educacionais baseados na teoria do construtivismo social.

2.1.6. Avaliação Heurística

Para testar a usabilidade de um software, são utilizadas técnicas de avaliação heurística, onde um grupo de avaliadores inspeciona a interação no software identificando problemas de usabilidade [Nielsen, 1994]. Baker et al. [2001] propõem um conjunto de heurísticas específicas para sistemas colaborativos. As heurísticas são: prover meios para a comunicação verbal e intencional; prover meios para a comunicação gestual e intencional; prover meios para transmitir informação não-intencional; prover meios para transmitir informação sobre as ações nos objetos compartilhados; prover proteção de acesso; gerenciamento de diferentes níveis de acoplamento no trabalho conjunto; possibilitar a coordenação; e facilitar encontros entre os participantes.

Araujo et al. [2004] propõem um ambiente para avaliação de protótipos de groupware. A avaliação é conduzida com base em quatro dimensões principais:

contexto do grupo, colaboração obtida enquanto o grupo trabalha, usabilidade do groupware e impactos culturais provocados na organização. Uma ontologia referente aos conceitos principais da avaliação guia o processo.

As abordagens apresentadas ao longo desta seção e a componentização de groupware não são excludentes. No Capítulo 6 é discutido como agregá-las e direcioná-las a uma engenharia de groupware baseada no modelo 3C de colaboração.

2.2. Componentes de Software

Nesta seção, são apresentados conceitos do desenvolvimento baseado em componentes (DBC), resumindo o Apêndice A. No DBC, componentes de software substituíveis, reusáveis e interoperáveis são utilizados para compor a aplicação final [Gimenes & Huzita, 2005]. Um componente de software é [D'Souza & Wills, 1998, p.387]:

> Um pacote coerente de software que (a) pode ser desenvolvido e instalado independentemente como uma unidade, (b) tem interfaces explícitas e bem definidas para os serviços que provê, (c) tem interfaces explícitas e bem definidas para os serviços que espera de outros, e (d) pode ser utilizado para composição com outros componentes, sem alterações em sua implementação, podendo eventualmente ser customizado em algumas de suas propriedades.

Um componente de software é instalado em uma plataforma de execução e segue um modelo de componentes [Szyperski, 1997]. É concebido para ser autocontido, reusável e substituível e prover serviços específicos de uma maneira coesa e bem definida. Os principiais benefícios da utilização de componentes são a manutenibilidade, reuso, composição, extensibilidade, integração, escalabilidade, entre outros [D'Souza & Wills, 1998, p.397].

Para acessar e interconectar componentes, são utilizadas suas portas [OMG, 2005]. Uma porta é um meio identificável de conexão, por onde um componente oferece seus serviços ou acessa os serviços dos outros [D'Souza & Wills, 1998, pg 410]. As portas são ligadas através de conectores, implementados através de chamada de métodos, propagação de eventos, fluxo de dados, transferência de arquivos, etc. [D'Souza & Wills, 1998, p.389]. Os tipos de conectores variam para

cada tecnologia e possibilitam a conexão em tempo de codificação, compilação, inicialização ou execução.

A interface é o contrato de utilização do componente [Szyperski, 1997]. Respeitando-se os contratos, pode-se alterar a implementação interna do componente ou substituí-lo por outro, sem modificar seus clientes. A interface define as maneiras de utilizar o componente, separando a especificação da implementação. Um componente apresenta múltiplas interfaces correspondendo aos conjuntos de serviços que visam diferentes necessidades dos clientes [D'Souza & Wills, 1998, p.397]. Normalmente, o componente possui pelo menos uma interface relativa aos serviços disponibilizados (interface de negócio) e outra à conexão com a infra-estrutura de execução (interface de sistema), onde são tratados serviços técnicos, como os relacionados ao ciclo de vida, à instalação e à persistência.

As interfaces são classificadas em fornecidas (*provided interfaces*) e requeridas (*required interfaces*) [Councill & Heineman, 2001, p.9]. Um componente possui uma interface fornecida ao implementar todas as operações definidas naquela interface e uma interface requerida ao usar pelo menos uma operação definida na interface. Na UML 2.0, interfaces fornecidas são representadas por uma circunferência fechada, enquanto as interfaces fornecidas são representadas por uma semicircunferência [OMG, 2005]. Conforme ilustrado na Figura 2.8, componentes se conectam por meio da interface requerida de um com a interface fornecida de outro [Barroca et al., 2005, p.6]. Para conectar componentes com conectores incompatíveis, desenvolve-se um código adicional chamado de adaptador, que faz as conversões e operações necessárias para compatibilizar interfaces.

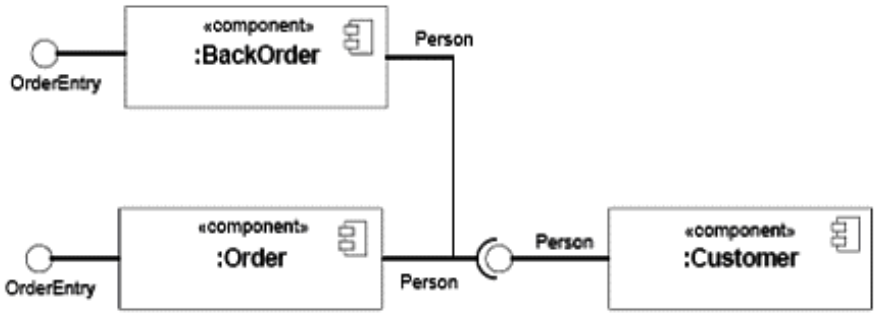

Figura 2.8. Exemplo de conexões entre componentes [OMG, 2005]

O modelo de componentes (*component model*) define vários aspectos da construção e da interação dos componentes, entre eles, a forma de implementar as interfaces e os conectores. Vários modelos apóiam-se na orientação a objetos para a implementação de interfaces e mensagens, entretanto, esta tecnologia não provê suporte à representação de interfaces requeridas e aspectos não-funcionais [D'Souza & Wills, 1998, p.388]. O modelo de componentes define também o padrão de nomeação dos componentes, de composição, de versionamento e de empacotamento. O empacotamento possibilita que um componente seja instalado como uma unidade, contendo arquivos, módulos, código executável, código fonte, código de validação, etc. [Szyperski, 1997, p.276; D'Souza & Wills, 1998, p.386].

Ao implantar o componente (*deployment*), ele é customizável [Heineman, 2000]. A customização é a habilidade de adaptar um componente antes de sua instalação ou uso, normalmente com o objetivo de especializar seu comportamento [Weinreich & Sametinger, 2001, p.42]. Na customização por composição, um componente repassa a outros as chamadas das operações. Na customização por alteração de propriedades, um arquivo descritor é utilizado para configurar o componente [Szyperski, 1997, p.33]. O arquivo descritor provê informações sobre o conteúdo do pacote, sobre as dependências externas e sobre as configurações do componente. Este arquivo é utilizado pela infra-estrutura de execução para instalar e configurar o componente [Weinreich & Sametinger, 2001, p.44].

Ao implementar um componente através da orientação a objetos, as interfaces definem o modelo de objetos compatível. Os objetos são passados de componente em componente, de modo que um componente não tem ciência da origem dos dados e do código sendo executado [Szyperski, 1997, p.42]. Chama-se de instância de componente, o conjunto de objetos pelos quais se manipula o componente [Szyperski, 1997, p.370]. Estes objetos são a manifestação do componente em tempo de execução [D'Souza & Wills, 1998, p.390].

Um *component kit* é um conjunto de componentes interoperáveis aderentes a uma padronização [D'Souza & Wills, 1998, p.404]. De um kit de componentes gera-se uma família de aplicações, fazendo diferentes arranjos e eventualmente desenvolvendo alguns sob medida [Wills, 2001, p.309]. Para desenvolver um

component kit, são analisadas aplicações similares e são identificados e generalizados componentes comuns [D'Souza & Wills, 1998, p.385].

Um *component framework* é um conjunto de interfaces e regras de interação que possibilitam a implantação de componentes aderentes a um certo padrão [Szyperski, 1997, p.26, p.280]. O *component framework* estabelece as "condições ambientais" e regula a interação entre as instâncias dos componentes.

Um container é uma plataforma, normalmente desenvolvida por terceiros, com o objetivo de hospedar e gerenciar componentes de um determinado modelo, provendo serviços de infra-estrutura de execução, como gerenciamento de transações distribuídas, pooling de recursos, acesso concorrente, segurança, persistência, etc. [D'Souza & Wills, 1998, p.401]. Os serviços providos pelo container e pelos *component frameworks* liberam os desenvolvedores de implementar serviços técnicos de baixo nível, de modo que direcionem seus esforços para as regras de negócio e para a composição do sistema. Alguns containeres possibilitam separar aspectos não relacionados à lógica da aplicação, possibilitando modificá-los sem alterar o componente. O container define uma interface que estabelece a conexão com os componentes, chamada de *lifecycle interface*, que é acessada pelo container para gerenciar a inicialização, execução e ativação do componente [Schwarz et al., 2003].

A arquitetura da aplicação descreve as propriedades, restrições e relacionamentos de suas partes [Stafford & Wolf, 2001, p.373]. O desenvolvimento baseado em componentes considera pelo menos duas visões da arquitetura: arquitetura de aplicação e arquitetura técnica [D'Souza & Wills, 1998, p.483]. Na arquitetura de aplicação são definidas a função de cada componente no contexto do sistema e a interação entre eles, de forma independente da tecnologia de suporte. A arquitetura técnica trata a tecnologia de suporte, independentemente do domínio da aplicação.

O conceito de frameworks está relacionado ao de componentes; são conceitos complementares que contribuem para o reuso de software [Gimenes & Huzita, 2005]. Os frameworks são voltados para um domínio específico de aplicação ou para a solução de problemas ligados à tecnologia. Um framework normalmente é orientado a objetos, composto de classes concretas e abstratas, interfaces e arquivos de configuração. Para especializar o framework utiliza-se

herança, composição ou configuração. Os pontos de extensão do framework são chamados de *hot-spots* [Johnson, 1997].

2.2.1. Benefícios e Dificuldades da Componentização de Software

Para projetar um sistema baseado em componentes é necessário entender os benefícios e dificuldades da tecnologia, além do ferramental disponível. Os benefícios da componentização estão ligados a manutenibilidade, reuso, composição, extensibilidade, integração, escalabilidade, entre outros [D'Souza & Wills, 1998, p.397]. As dificuldades são separadas em dificuldades do desenvolvimento para componentização (construção dos componentes e da infra-estrutura) e dificuldades do desenvolvimento com componentização. As dificuldades do desenvolvimento para componentização estão ligadas ao esforço inicial de análise, projeto e desenvolvimento, enquanto as dificuldades do desenvolvimento com componentização estão ligadas ao esforço despendido no entendimento dos componentes e das ferramentas envolvidas, à perda de flexibilidade, à dependência de terceiros e à adaptação do processo de desenvolvimento.

Uma das principais motivações para se desenvolver um software baseado em componentes é sua manutenção. Os componentes são substituíveis para atualização ou correção, muitas vezes sem precisar alterar ou recompilar a aplicação como um todo [D'Souza & Wills, 1998]. Componentes são adicionados, removidos ou substituídos por versões mais robustas ou mais apropriadas ao hardware, ao sistema operacional ou aos produtos legados com os quais o sistema tenha que operar. A modularidade obtida com a componentização facilita a localização do código a ser alterado e o encapsulamento da alteração. Algumas mudanças não acarretam em alterações no código, sendo resolvidas por re-composição da aplicação ou re-configuração dos componentes [D'Souza & Wills, 1998, p.397].

O reuso também é freqüentemente citado como um benefício da componentização. O reuso favorece a redução dos esforços de desenvolvimento e a qualidade do produto final, por colocar em uso código já utilizado e testado em

outras situações [Krueger, 1992]. Blocos com granularidade baixa (como uma classe) e alta (como um sistema) são difíceis de reusar, pois são genéricos ou específicos demais. O desenvolvimento baseado em componentes trabalha com blocos com granularidade média, mais propícia para o reuso. A componentização também promove o reuso nas diversas atividades do desenvolvimento: análise, projeto, implementação e testes.

Com a componentização, a aplicação é adaptável para diversas necessidades, selecionando e configurando os componentes mais adequados. Pode-se reimplementar um determinado componente para atender a uma necessidade específica ou adicionar novos componentes ou interfaces para estender os serviços providos, tornando o software desenvolvido mais adaptável e extensível [D'Souza & Wills, 1998, p.397].

Uma outra vantagem da componentização é o encapsulamento de conhecimento e uma programação de alto nível. Um desenvolvedor não precisa conhecer os detalhes de implementação dos componentes para utilizá-los para compor as aplicações. Quem integra componentes se especializa nesta atividade abstraindo os detalhes de implementação, tendo uma visão mais abrangente e mais próxima do domínio de aplicação.

A componentização também facilita a prototipação, pois o sistema é recomposto para experimentar idéias. Esta capacidade é especialmente útil quando há pressão para liberar produtos no mercado ou em sistemas com requisitos não definidos e instáveis [Pressman, 2000]. Com a componentização, é possível usar um ambiente RAD (*Rapid Application Development*) ou um toolkit para o desenvolvedor prototipar sua aplicação. A prototipação e o desenvolvimento iterativo possibilitam colocar o sistema em produção mais cedo, de modo a refinar gradualmente os requisitos e construir o sistema com base no aprendizado obtido com a realimentação [Teles, 2004].

A decomposição do sistema possibilita a definição de componentes independentes, que podem ser subcontratados ou alocados para outras equipes, o que favorece o desenvolvimento paralelo e em grupo. Em alguns sistemas [Won et al., 2005; Slagter & Biemans, 2000; Li & Muntz, 1998; Hummes & Merialdo, 2000], os próprios usuários finais recompõem e re-configuram os componentes, adaptando a aplicação para suas necessidades específicas. A aplicação é

incrementada para acompanhar as características das tarefas e para prototipar e experimentar configurações antes de solicitar um desenvolvimento completo.

O desenvolvedor de um componente é beneficiado pela infra-estrutura de execução, que provê serviços básicos como persistência, interconexão, escalabilidade, etc., aliviando a necessidade de implementar estes serviços. Algumas infra-estruturas possibilitam a integração de forma transparente para o programador de componentes desenvolvidos e disponibilizados em diferentes linguagens de programação, tecnologias e plataformas [D'Souza & Wills, 1998, p.397].

Com relação às dificuldades, o desenvolvimento baseado em componentes demanda um esforço inicial maior de projeto e implementação para montar a infra-estrutura do sistema e construir uma biblioteca robusta de componentes reusáveis. Projetar e preparar um pedaço de software para futuro reuso aumenta a necessidade de flexibilidade, documentação, estabilidade e abrangência do software [Moore & Bailin, 1991]. O software deve ser bem documentado, testado e deve ter um esquema robusto de validação [Pfleeger, 2001]. Os componentes não podem ser nem muito genéricos e nem muito específicos [Oliveira, 2001].

O custo do estudo e entendimento de como usar e instalar os componentes também é outra dificuldade associada com o reuso. Às vezes é mais rápido desenvolver um componente do que procurar por um pronto, estudá-lo e adaptá-lo [Pfleeger, 2001]. A menos que o custo de aprendizagem seja amortizado por vários projetos ou que o ganho de produtividade e qualidade sejam expressivos, o investimento inicial não se torna atraente [Oliveira, 2001].

O reuso de componentes provenientes de terceiros pode levar a situações inesperadas, onde se torna necessário reimplementar uma grande quantidade de código, quando novos requisitos levarem a alterações ou customizações não possibilitadas pelo componente. Os componentes introduzem dependências fora do controle dos desenvolvedores do sistema, impondo um esforço continuado de atualização de suas versões e de reconfiguração do sistema. Há um risco de incorporar bugs de terceiros no software. Além disto, muitas vezes é difícil encontrar um componente que atenda plenamente às funcionalidades desejadas e ainda ofereça suporte aos requisitos não funcionais da aplicação, como performance, segurança, escalabilidade, etc. [Gimenes & Huzita, 2005]. Por fim, o

desenvolvimento baseado em componentes também demanda uma adaptação no processo de desenvolvimento, que passa a incluir etapas como análise de domínio, busca de componentes e testes específicos.

A tecnologia de componentes vem sendo constantemente utilizada no desenvolvimento de groupware, conforme é ilustrado na próxima seção. A manutenibilidade, reuso, composição, extensibilidade, integração e escalabilidade obtidos com a componentização normalmente superam as dificuldades advindas da tecnologia e são importantes no desenvolvimento de sistemas colaborativos. As dificuldades da componentização aplicadas à abordagem e à arquitetura propostas nesta tese são retomadas no Capítulo 5.

2.3. Groupware Baseado em Componentes

Na literatura, a tecnologia de componentes é vista como apropriada para o desenvolvimento de groupware e há diversos sistemas e plataformas que disponibilizam blocos modulares e relativamente independentes para a construção das aplicações colaborativas. Os sistemas baseados em componentes são classificados em estritamente baseado em componentes (*strictly component-based*) ou fracamente baseado em componentes (*loosely component-based*) [Slagter & Hofte, 1999]. O primeiro contempla os sistemas onde um modelo de componentes é utilizado e o segundo enfoca sistemas onde os componentes são vistos como partes independentes, sem uma padronização específica. A utilização de um modelo de componentes favorece a integração, o reuso e a substitutibilidade dos componentes, bem como a possibilidade de extensão da solução por terceiros [D'Souza & Wills, 1998]. Por sua maior relação com a proposta desta tese, na seqüência são descritos sucintamente alguns sistemas de groupware estritamente baseados em componentes.

2.3.1. Live

Live [Banavar et al., 1998] é uma plataforma que oferece suporte a construção de groupware síncrono. O modelo de componentes do LIVE é baseado

na especificação JavaBeans e fornece uma interface de alto nível para o desenvolvimento de groupware. Os componentes do Live são baseados nos conceitos de sessões e objetos. Os objetos são compartilhados no contexto de uma sessão, que é implementada no servidor. Quatro tipos de objetos compartilhados são disponibilizados: *stateful objects*, que mantém o estado persistente; *media stream objects*, que são utilizados para prover a transmissão de áudio e vídeo no modo contínuo; *stateless objects*, que são utilizados para prover notificação de eventos e sincronização; e *token objects*, que são utilizados para prover controle de concorrência. A plataforma Live oferece serviços específicos para cada um dos tipos de objetos.

Os componentes disponibilizados na plataforma Live são predominantemente não-visuais, ou seja, não é provida uma interface com usuário para estes componentes. Entretanto, a plataforma provê funcionalidades específicas para a integração de componentes visuais provenientes de terceiros. Alguns exemplos de funcionalidades providas pelos componentes da plataforma Live são: gerenciamento de sessão, gerenciamento de convites, compartilhamento de dados e eventos, sincronização de acesso, transmissão contínua e arquivamento e recuperação de objetos e eventos.

A diferença fundamental entre os componentes da plataforma Live e os componentes propostos nesta tese está na concepção e enfoque. Os componentes desta tese são derivados de uma engenharia do domínio guiada pelo modelo 3C de colaboração, que apóia as diversas etapas do ciclo de desenvolvimento de groupware. O enfoque dos componentes da plataforma Live é em groupware síncrono, com grande parte das funcionalidades voltadas para transmissão contínua de áudio e vídeo. Os componentes desta tese, assim como os componentes da plataforma Live, foram propostos com o intuito de encapsular a complexidade de baixo nível inerente a sistemas colaborativos, com o objetivo de reduzir as dificuldades de desenvolvimento de groupware.

2.3.2. DISCIPLINE

DISCIPLINE (DIstributed *System for Collaborative Information Processing and LEarning*) [Marsic, 1999] é uma plataforma voltada para o desenvolvimento de groupware síncrono para o domínio educacional. A arquitetura do DISCIPLINE consiste de componentes replicados e recursos centralizados no servidor. Os elementos principais desta arquitetura são um barramento de colaboração, uma interface com usuário multi-modal, lógica da aplicação e agentes inteligentes.

O barramento de colaboração interconecta os elementos da arquitetura, propagando eventos e controlando a concorrência de acesso e a sincronização entre as aplicações, bem como o gerenciamento das informações de percepção. A interface com o usuário multi-modal provê diversas formas de interação entre os usuários e a aplicação. Alguns recursos oferecidos, além dos recursos de entrada e de saída textual, são o reconhecimento e a sintetização de voz e o reconhecimento de gestos. Recentemente, também foi incorporado na plataforma o suporte a construção de interfaces para dispositivos móveis, como PDAs [Krebs et al., 2003]. A lógica da aplicação é provida por componentes JavaBeans, e agentes inteligentes são utilizados para coordenar os elementos da arquitetura.

O enfoque da plataforma DISCIPLINE é no desenvolvimento de groupware síncrono com integração entre diversas ferramentas e mecanismos de entrada e de saída. Nesta tese, a abordagem proposta é direcionada para o modelo de negócio da aplicação, que é organizada em função do modelo 3C. Com relação à interface com o usuário, o suporte computacional à colaboração é harmonicamente combinado de modo a facilitar a manipulação das ferramentas e informações [Laurillau & Nigay, 2002].

2.3.3. FreEvolve

O FreEvolve [Won et al., 2005], chamado anteriormente de EVOLVE [Stiemerling et al., 1999] (antes de sua liberação na licença GPL), é um sistema baseado em componentes desenvolvido em uma arquitetura cliente-servidor na Internet. O FreEvolve foi concebido visando a adaptação e a montagem pelos usuários finais da aplicação. O modelo de componentes utilizado no FreEvolve é chamado FlexiBeans, e é uma extensão do JavaBeans. O suporte específico a portas, objetos compartilhados e interações remotas é parte das extensões do FlexiBeans. O modelo de componentes contempla tanto componentes visíveis na interface com o usuário, quanto componentes sem apresentação. No FreEvolve, a estrutura da aplicação colaborativa é descrita por arquivos que descrevem a composição da estrutura no lado servidor e no lado cliente, as interconexões entre os componentes e a ligação entre as ferramentas e os usuários. A composição pelo usuário final é alcançada a partir da manipulação destes arquivos.

O sistema FreEvolve oferece também uma API (Tailoring API) que disponibiliza recursos de customização e composição. Posteriormente, foi incorporada também no sistema uma interface gráfica 3D para manipulação e configuração dos componentes [Stiemerling et al., 2001]. O cliente do sistema FreEvolve pode ser uma aplicação stand-alone ou embutida em um navegador web. A versão embutida possui uma instalação mais direta, porém a versão stand-alone possui mais funcionalidades, visto que não está sujeita às mesmas restrições de segurança presentes na versão para navegadores.

O enfoque do FreEvolve é na geração de groupware extensível, apto a acompanhar a evolução dos processos de trabalho. A abordagem do FreEvolve é similar à arquitetura proposta nesta tese no aspecto de encapsular a complexidade da montagem da aplicação colaborativa através da utilização de arquivos para manipulação dos componentes do sistema. O uso de arquivos explicita a estrutura e a configuração do sistema e dispensa a utilização de uma ferramenta específica para a composição e customização do ambiente. Entretanto, assim como nas plataformas anteriores, o FreEvolve não utiliza explicitamente um modelo de colaboração para a concepção dos componentes e para a montagem do ambiente colaborativo.

2.3.4. DACIA

A plataforma DACIA (*Dynamic Adjustment of Component InterActions*) [Litiu & Prakash, 2000] é voltada para o desenvolvimento de groupware para dispositivos móveis. A plataforma define seu próprio modelo de componentes. Neste modelo, um componente é chamado de PROC (Processing and ROuting Component). O modelo de componentes define uma maneira particular de troca de mensagens através das portas de entrada e de saída dos componentes. Os componentes transformam os fluxos de entrada, sincronizam os fluxos de entrada e de saída e, eventualmente, direcionam partes dos fluxos de entrada para múltiplos destinos.

Na arquitetura DACIA, há um elemento chamado Engine que é executado em cada estação e é responsável pelo gerenciamento dos componentes, mantendo uma lista dos PROCs e de suas conexões. Este elemento é responsável por estabelecer e manter as conexões e as comunicações entre as estações e seus componentes. A arquitetura DACIA também prevê o uso de monitores. Um monitor trabalha em conjunto com um Engine e é responsável pelo gerenciamento dos dados e pela configuração dos componentes. Enquanto os Engines e os PROCs são de propósito geral, os monitores são específicos da aplicação em questão.

Os princípios de projeto da plataforma DACIA são a obtenção de uma arquitetura modular, usada para construção de aplicações a partir de componentes de software que implementam operações individuais; o oferecimento de mobilidade para a aplicação e para o usuário; o encapsulamento das complexidades inerentes ao desenvolvimento de aplicações para dispositivos móveis, como o gerenciamento de uma conexão intermitente; o oferecimento de suporte à conexão entre componentes remotos; e a possibilidade de reconfiguração da aplicação em tempo de execução.

Atualmente, considerar os dispositivos móveis no desenvolvimento de groupware é fundamental, dada a disseminação de PDAs e celulares com capacidade de processamento. A partir de 2004 foi dado início ao

desenvolvimento do AulaNetM, uma extensão do AulaNet para equipamentos móveis [Filippo et al., 2005]. O enfoque da arquitetura e da componentização propostas nesta tese é voltado para a camada de negócios da aplicação, prevendo a possibilidade de mais de um meio de interação. Deste modo, a lógica do suporte computacional à colaboração é reusada para uma camada de visão que funciona através da web, de aplicação stand-alone ou de dispositivos móveis. Diferentemente dos componentes previstos na arquitetura DACIA, os componentes desta tese são desenvolvidos e organizados seguindo uma abordagem 3C.

2.3.5. DreamTeam

O DreamTeam [Roth & Unger, 2000] é uma plataforma baseada em componentes para a montagem de groupware síncrono. A plataforma disponibiliza um ambiente de desenvolvimento, com ferramentas específicas, um ambiente de execução e um ambiente de simulação. São disponibilizados ao desenvolvedor componentes de groupware, componentes de interface com o usuário e componentes de manipulação de dados. Os componentes são chamados de TeamComponents e são associados aos componentes de interface com o usuário e de manipulação de dados. Os componentes TeamComponents são desenvolvidos em Java, porém seguem um modelo de componentes proprietário.

Na arquitetura do DreamTeam, os componentes não se comunicam diretamente uns com os outros. As mensagens são roteadas através de uma interface da aplicação. O modo de comunicação pode ser “intra-site”, onde um componente se comunica com outro da mesma aplicação, e “inter-site” onde um componente comunica-se com componentes de outra aplicação. O primeiro modo utiliza mensagens Java e o segundo, um mecanismo próprio de propagação de eventos. A interface dos componentes provê métodos para acessar os atributos do componente, como sua configuração, modo de integração e estado. Há também funcionalidades específicas para notificação de eventos.

Os componentes são integráveis dinamicamente à aplicação. A interface com o usuário do componente se torna parte da interface da aplicação em uma

janela existente ou como uma nova janela. O componente é configurável para diferentes modos de colaboração: privado, exclusivo ou compartilhado. Um componente sendo executado no modo privado não possui dados compartilhados; no modo exclusivo, os dados são modificáveis por um usuário de cada vez; e no modo compartilhado, vários usuários manipulam simultaneamente os dados. O DreamTeam possui diversos mecanismos para controle de concorrência, como o gerenciamento de transações envolvendo a aplicação, os recursos e a definição de políticas de acesso para os componentes e seus métodos.

O enfoque do modelo de componentes do DreamTeam é o suporte à conexão entre componentes distribuídos, de uma maneira particular para groupware. Na arquitetura proposta nesta tese não há a preocupação com a distribuição dos componentes, pois a maior parte deles é executada no servidor. Quando a comunicação remota é necessária, são utilizados os mecanismos padrões da plataforma J2EE (http://java.sun.com/j2ee).

2.3.6. IRIS

IRIS [Koch & Koch, 2000] é uma aplicação para edição colaborativa de documentos multimídia, construída a partir de componentes que utilizam um mecanismo de persistência distribuído e serviços voltados ao gerenciamento de informações de percepção. O IRIS oferece funcionalidades para editar e exibir a estrutura e o conteúdo de documentos, e para contextualizar o participante através de informações, como quem está trabalhando com o documento, quem está disponível, etc.

Os componentes da aplicação são chamados de *tools*. Para oferecer um ambiente de execução aos componentes, é disponibilizado na plataforma IRIS um *component framework* chamado ICI (IRIS Component Infrastructure). Este *component framework* possibilita a integração de diferentes ferramentas e oferece serviços de execução, que facilitam o desenvolvimento de novos componentes para a plataforma. O ICI possui um gerente de componentes chamado ToolConnector que gerencia as ferramentas e oferece a elas serviços de execução.

Na arquitetura do IRIS não há componentes centrais. As ferramentas, os dados e o controle são replicados. Eventualmente, um usuário trabalha desconectado no documento compartilhado e, ao sincronizar os dados, a plataforma IRIS oferece mecanismos para identificar e tratar os eventuais conflitos e gerenciar as versões intermediárias. A plataforma também oferece recursos para importação e exportação dos dados.

O foco do IRIS é a investigação das questões de consistência e integridade em um ambiente multi-usuário distribuído, onde os participantes trabalham conectados e desconectados, sincronamente e assincronamente. O IRIS foi desenvolvido também com o intuito de investigar as informações de percepção necessárias para o trabalho em grupo. Diferentemente do IRIS, nesta tese não há uma preocupação explícita com a autoria compartilhada de documentos e o suporte computacional à percepção é distribuído no suporte computacional à comunicação, coordenação e cooperação.

2.3.7. JViews

JViews [Grundy et al., 1997] é um *component framework* voltado para a construção de sistemas colaborativos com múltiplas visões. É disponibilizado um repositório central, que é encarregado de propagar as alterações para as visões afetadas, seguindo o padrão modelo, visão e controle. As inter-relações entre os componentes são utilizadas para troca de dados, agregação de funcionalidades e mantenção da consistência geral do sistema.

O JViews também disponibiliza um modelo de eventos que é utilizado para descrever as mudanças nos componentes ou ocorrências que são tratadas externamente. Para manter a consistência entre os diversos componentes e visões, o JViews disponibiliza políticas de restrições de integridade, mecanismos de persistência, meios de propagação de eventos, componentes adaptadores, controle de versões e registro dos estados e eventos, de modo a oferecer suporte às operações de desfazer e refazer ações.

Nesta tese, as funcionalidades presentes no JViews são tratadas pelos frameworks de infra-estrutura, responsáveis pela persistência, gerenciamento dos

dados, propagação de eventos, transações distribuídas, etc. Mais detalhes da arquitetura técnica utilizada nesta tese são descritos na dissertação de Barreto [2006].

2.3.8. CoCoWare

A plataforma CoCoWare [Slagter & Biemans, 2000] oferece aos usuários finais a capacidade de compor a aplicação de acordo com suas necessidades e estendê-la para acompanhar a evolução dos processos de trabalho. O CoCoWare oferece componentes para lidar com as sessões de trabalho e para gerenciar as ferramentas colaborativas, informando o que pode ser modificado, como pode ser feito e qual o impacto das modificações. O CoCoWare é uma implementação da arquitetura genérica CooPS (Cooperative People & Systems), que define os tipos básicos de componentes para a construção de aplicações colaborativas: *conference manager*, *conference tools*, *coordinator* e *conference enablers* [Slagter et al., 2001].

A plataforma foi inicialmente baseada no modelo de componentes do CORBA e, atualmente, estende o modelo de componentes da arquitetura .NET da Microsoft. O CoCoWare oferece um *component framework* onde os componentes são plugados e é disponibilizado um Software Development Kit (SDK) para terceiros desenvolverem novos componentes para a plataforma ou adaptarem componentes de outros modelos (COM e ActiveX).

Diferentemente do enfoque principal do CoCoWare, a abordagem proposta nesta tese visa instrumentar o desenvolvedor de groupware. Entretanto, alguma capacidade de extensão é oferecida ao administrador do ambiente, que pode instalar novos componentes e configurá-los.

2.3.9. Habanero

O Habanero [Chabert et al., 1998] provê aos desenvolvedores ferramentas para criar aplicações colaborativas em Java. É oferecido um ferramental para criar ou migrar aplicações e applets existentes para o ambiente de groupware. O

Habanero disponibiliza um servidor que hospeda e gerencia as sessões e as conexões dos clientes. O *component framework* da plataforma disponibiliza serviços de registro de sessões, gerenciamento de participantes e controle de acesso. As aplicações clientes do Habanero são chamadas de Hablets. Através da interface provida pelo *component framework*, um Hablet cria, conecta e visualiza informações das sessões e dos participantes, gerencia e propaga eventos, ativa ferramentas e manipula o catálogo de endereços compartilhado. Alguns exemplos de ferramentas desenvolvidas para a plataforma Habanero são um navegador web compartilhado, um whiteboard, um bate papo em áudio, um visualizador cooperativo de documentos VRML e um compartilhador de área de trabalho, baseado no VNC (*Virtual Networking Computing*).

As ferramentas são replicadas nos clientes e as mudanças de estados são distribuídas. Quando um novo cliente entra em uma sessão, são enviadas informações de quais ferramentas estão ativas naquela sessão e os dados necessários para compor o estado atual da colaboração. O Habanero possibilita múltiplas sessões, sendo que um mesmo participante pode atuar em mais de uma sessão simultaneamente. Uma nova ferramenta é integrada dinamicamente a uma sessão existente. A plataforma possibilita também salvar o estado de uma sessão de modo a prosseguir nela posteriormente. No Habanero não há o conceito explícito de um modelo de colaboração. Nesta plataforma, os elementos do suporte computacional à colaboração são associados ao gerenciamento de sessões.

2.3.10. COCA

A plataforma COCA (*Collaborative Objects Coordination Architecture*) [Li & Muntz, 1998] oferece uma maneira de separar as políticas de coordenação, que normalmente ficam embutidas no código da aplicação. As políticas de coordenação são descritas em uma linguagem de definição, e o coordenador do grupo pode alterar dinamicamente estas políticas, refinando o ambiente para acompanhar a evolução da realização das tarefas. As políticas são definidas em função dos papéis dos usuários. A máquina virtual que interpreta as políticas de coordenação, chamada de *Cocavm*, é replicada nas estações dos clientes.

A arquitetura do COCA prevê uma organização em camadas, com uma camada de comunicação, de coordenação e de colaboração. A camada de comunicação oferece serviços de transmissão de dados para todas estações ou para um determinado usuário em particular. Acima desta camada, vem a camada de coordenação que interpreta as políticas de coordenação. A camada de colaboração é responsável por oferecer uma infra-estrutura de execução para as ferramentas colaborativas. A arquitetura também provê um barramento de colaboração que abstrai as funcionalidades do canal de comunicação.

Apesar da arquitetura do COCA utilizar comunicação, coordenação e colaboração, o entendimento destes conceitos difere do modelo 3C utilizado nesta tese. A camada de comunicação do COCA é voltada para a comunicação remota entre componentes em vez de participantes. A arquitetura do COCA não oferece suporte específico ao conceito de cooperação do modelo 3C. A camada de colaboração funciona como um *component framework* para os componentes de colaboração. Além disto, os conceitos adotados na arquitetura do COCA não são utilizados para organizar ou conceber os componentes de colaboração.

2.3.11. GroupKit

O GroupKit [Roseman & Greenberg, 1996] é um toolkit contendo componentes e uma plataforma de execução. O GroupKit é construído em Tcl/Tk e é voltado ao desenvolvimento de groupware síncrono. O toolkit encapsula diversas complexidades inerentes ao desenvolvimento deste tipo de aplicação, de modo que o desenvolvedor direcione seus esforços para o projeto da interação. Alguns exemplos de aplicações desenvolvidas com o kit, que são disponibilizadas juntamente com a plataforma, são: Brainstorming, que possibilita a usuários trocar idéias através de textos breves em uma área visível a todos; File Viewer, que possibilita que vários usuários visualizem um texto simultaneamente; Hello World, que disponibiliza um botão que quando pressionado, transmite uma mensagem para todos usuários conectados; Text Chat, um programa de chat semelhante ao Talk do ambiente Unix, que exibe imediatamente o que cada pessoa escreve; Tic Tac Toe, o Jogo da Velha; e Tetrominoes, para rodar e mover

diversos tipos de polígonos. A Figura 2.9 apresenta algumas aplicações desenvolvidas utilizando o GroupKit.

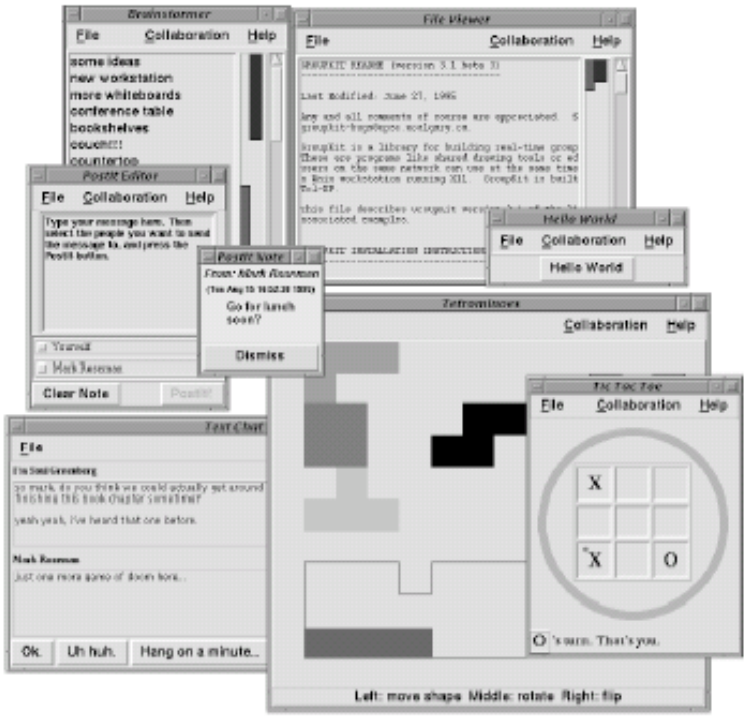

Figura 2.9. Aplicações desenvolvidas utilizando o GroupKit

O ambiente de execução do GroupKit gerencia a criação, a interconexão e a comunicação dos processos distribuídos que compõe a sessão colaborativa. O GroupKit oferece facilidades aos programadores para interconexão, gerenciamento de eventos e compartilhamento de dados. Cada estação cliente possui uma réplica do gerenciamento de sessões e da ferramenta colaborativa, enquanto o repositório fica localizado em um servidor central. O gerenciador de sessões de cada participante transmite informações sobre a sessão e sobre os participantes para o repositório central. Após o estabelecimento da conexão inicial, as ferramentas trocam informações diretamente, através de chamadas remotas de procedimento, de propagação de eventos ou de variáveis de ambiente compartilhadas entre as diversas aplicações. São oferecidos aos programadores mecanismos que encapsulam e abstraem os detalhes técnicos de conexão e comunicação.

O GroupKit oferece diversos widgets de interface que são utilizados para compor a interface gráfica da aplicação. O modelo de componentes adotado possibilita que o desenvolvedor crie novos ou estenda os widgets existentes. Há

widgets para prover informações de percepção, telepointers, barras de rolagem multi-usuário, visão de radar da área compartilhada, etc. Seus widgets são particularmente relevantes para a construção de interfaces WYSIWS (*What You See Is What I See*) relaxadas, onde as dimensões das janelas utilizadas podem ser distintas. O GroupKit foi utilizado como base para a construção de diversos sistemas colaborativos, como o Alliance [Michailidis & Rada, 1996] e o GroupCRC [Churcher & Cerecke, 1996]. Recentemente, o GroupLab[1], que desenvolve o GroupKit, disponibilizou também widgets voltados a encapsular os detalhes técnicos de interação com dispositivos físicos (phidgets), de modo que o desenvolvedor monta uma solução integrada de hardware e software [Greenberg & Fitchett, 2001]. Também foi disponibilizado um toolkit específico para SDG (*Single Display Groupware*), que objetiva oferecer suporte computacional à colaboração de vários participantes utilizando uma mesma máquina, com vários dispositivos de entrada e saída [Tse & Greenberg, 2004].

O GroupKit possui um enfoque maior na construção da interface com o usuário para groupware síncrono. Nesta tese, o enfoque é em instrumentar o desenvolvedor oferecendo componentes organizados em função do modelo 3C para construção de sistemas colaborativos na web. Apesar de o estudo de caso feito com o ambiente AulaNet possuir widgets de interface, sua construção foge do escopo desta tese.

2.3.12. Portalware

Portalware, também conhecido como CMS (*Content Management System*), é um tipo de sistema utilizado para construir portais na web. Estes sistemas apresentam uma arquitetura modular baseada em componentes para a construção de portais que oferecem suporte à interação com os usuários. Alguns exemplos de portalware são o Lumis[2], o Mambo[3], o XOOPS[4], o Zope[5] e o DotNetNuke[6]. Estes

[1] http://grouplab.cpsc.ucalgary.ca
[2] http://www.lumis.com.br
[3] http://www.mamboserver.com
[4] http://www.xoops.org
[5] http://www.zope.org
[6] http://www.dotnetnuke.com

sistemas oferecem flexibilidade para o administrador do portal compor as páginas incluindo ferramentas, configurando-as e posicionando-as.

Figura 2.10. Seleção de serviços no Lumis

A Figura 2.10 apresenta a tela de seleção de serviços do Lumis. Alguns dos serviços disponíveis neste ambiente são chat, e-mail, fórum de discussão, boletim, agenda de grupo e pessoal, relatórios e estatísticas, tarefas, gerenciamento de usuários e grupos, enquete, alertas, artigos, documentos, matérias e comentários. Cada serviço responde às requisições com dados no formato XML, que são convertidos para o formato HTML através de scripts XSL. Esta conversão possibilita a utilização do mesmo serviço em diversos portais ou em diferentes seções do mesmo portal, com diferentes padronizações visuais, a utilização do mesmo serviço em várias plataformas, e a criação de *skins*, que possibilitam alterar a interface do ambiente como um todo.

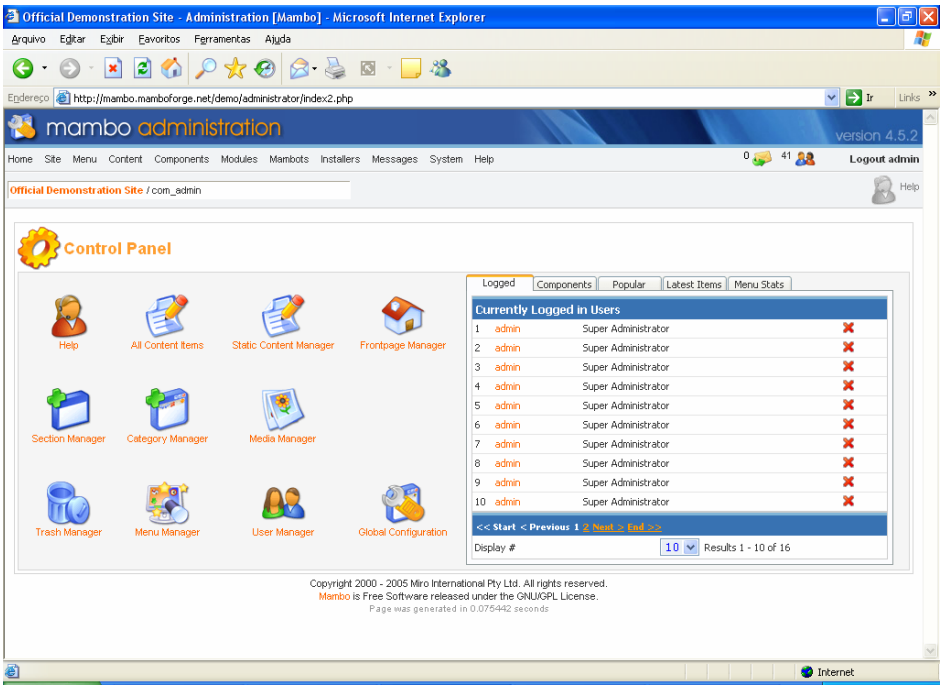

Figura 2.11. Interface administrativa do Mambo

O Mambo, cuja interface administrativa é apresentada na Figura 2.11, é desenvolvido na linguagem PHP e possui código fonte aberto. As ferramentas de colaboração do Mambo são tratadas como componentes e há um modelo de componentes que padroniza a construção de novos módulos ao ambiente. Da mesma maneira que o Lumis, o Mambo possibilita ao administrador montar as páginas, adicionando e configurando os serviços, e oferece a possibilidade de uso de *skins*. O Mambo oferece diversas opções de configuração para as páginas do portal, como tamanho, posicionamento de *banners* e estilos. Grande parte dos serviços disponíveis na plataforma é voltada para o gerenciamento de conteúdos, como documentos, matérias e notícias.

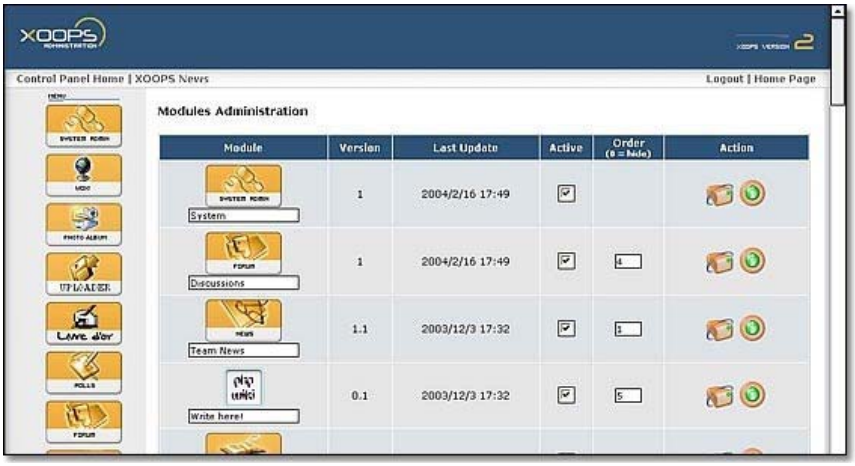

Figura 2.12. Gerenciamento de serviços no XOOPS

O XOOPS (*eXtensible Object Oriented Portal System*) também foi desenvolvido na linguagem PHP e está disponível sob a licença GNU General Public License (GPL). O site é customizado pelo administrador, que adiciona ou remove módulos através do ambiente administrativo. A Figura 2.12 apresenta a tela de gerenciamento de serviços da plataforma. Cada módulo do XOOPS oferece um conjunto de blocos pré-definidos para exibição das saídas, que são utilizados para compor a interface com o usuário. Alguns módulos já disponíveis na instalação do XOOPS são Notícias, Fórum, Enquete, Links, Downloads, Headlines, FAQ, Parceiros (banners), Usuários e Fale Conosco.

O DotNetNuke é desenvolvido para a plataforma .NET da Microsoft. Alguns serviços disponíveis na plataforma são Avisos, Banners, Newsfeeds, Fórum, FAQ, Lista de discussão, Calendário, Links, Busca e Enquetes. Assim como os demais portalware, o DotNetNuke possibilita o uso do *skins* para troca da aparência do ambiente como um todo.

A Sun Microsystems, através do Java Community Process (JCP), disponibilizou uma especificação para padronizar componentes e containeres para portais. De acordo com o modelo, os *portlets*, como são chamados os componentes, geram fragmentos de páginas que são compostas pelo container para gerar a página final. O container intermedeia a interação entre clientes e *portlets* e gerencia seu ciclo de vida. Mais detalhes sobre o modelo de componentes do *portlets* são encontrados no Apêndice A.

Nenhuma das tecnologias para construção de portais analisadas apresenta um modelo de componentes que incorpora aspectos específicos do suporte computacional à colaboração. Os modelos de componentes destas plataformas oferecem recursos para encapsular serviços técnicos e para cuidar do ciclo de vida dos componentes. Até o momento não há um padrão único para a construção de serviços para portais. Pretende-se futuramente desenvolver adaptadores do modelo de componentes utilizado nesta tese para as infra-estruturas de execução dos portalware, de modo a compatibilizar os serviços desenvolvidos para o ambiente AulaNet com os portalware e vice-versa.

2.3.13. Outras Plataformas para a Construção de Groupware

A plataforma ACOST [Hummes & Merialdo, 2000] é voltada para a criação de sistemas colaborativos extensíveis pelos usuários em tempo de execução. O modelo de componentes JavaBeans é utilizado como base para o modelo de componentes definido na plataforma. A plataforma estende o modelo de eventos definido no JavaBeans para possibilitar a troca de eventos entre componentes, mesmo eles estando distribuídos na rede. O ACOST é primordialmente utilizado para a construção de ferramentas de comunicação síncrona. Alguns exemplos de ferramentas construídas utilizando a plataforma incluem um chat e um sistema de votação online.

O Flexible JAMM (*Java Applets Made Multiuser*) [Begole et al., 1999] provê componentes alternativos para a biblioteca gráfica do Java para transformar um applet mono-usuário em uma aplicação multi-usuário, sem alterar a aplicação original. Os componentes possibilitam que diversos usuários visualizem e editem as informações, além de oferecer telepointer, visão de radar, controle de permissões e indicação de atividade. As versões mono-usuário dos componentes gráficos são substituídas dinamicamente por versões colaborativas. A substituição dos componentes é feita por uma customização na máquina virtual Java, o que reduz a sua portabilidade.

TOP [Guerrero & Fuller, 1999] é um framework em Java voltado para o desenvolvimento de aplicações colaborativas na web. Ele provê abstrações pré-definidas, que possibilitam o gerenciamento dos usuários e seus papéis, a manutenção da memória do grupo e os relacionamentos entre os participantes das sessões colaborativas. Um applet intermedeia a comunicação do cliente com o servidor, chamado de TopServer. O applet é acessível através de comandos JavaScript ou de outras aplicações Java.

Sacramento et al. [2004] propõem o middleware MoCA (*Mobile Collaboration Architecture*) voltado para oferecer suporte ao desenvolvimento de aplicações colaborativas que utilizam dispositivos móveis e informações de contexto. O MoCA oferece APIs para programação de aplicações para o lado cliente e para o lado servidor, serviços para monitoramento e inferência da

localização e contexto dos dispositivos e um framework orientado a objetos para instanciar *proxies* customizados. O middleware encapsula funcionalidades para lidar com a mobilidade do dispositivo, com a limitação de recursos e com a intermitência da conexão.

Siqueira et al. [2003] propõem uma arquitetura baseada em frameworks e componentes que possibilita instanciar ambientes educacionais para diferentes metodologias e teorias de aprendizagem, de forma a moldar um ambiente específico para cada caso. Os componentes principais da arquitetura são: *Data* e *Metadata Management, Groupware Management, Content Development Management, Assessment and Evaluation Management, Interface Management, Role and Security Management* e *Rule Management*.

O sistema Sieve [Isenhour et al., 1997] provê suporte à montagem de aplicações para visualização colaborativa de dados. Os componentes são interligados de modo a conectar fontes de dados e componentes de processamento e de visualização. O modelo de componentes do Sieve é uma extensão do JavaBeans. A aplicação é extensível dinamicamente através da inserção de novos componentes. Os dados são processados em um servidor central, de modo que todos os participantes visualizam o mesmo fluxo, em um estilo WYSIWIS (*What You See Is What I See*). Entretanto, os participantes podem atuar em diferentes partes das informações.

Dourish [1998] propõe a plataforma Prospero, que visa oferecer mais flexibilidade que os toolkits tradicionais, ao oferecer uma meta-arquitetura utilizada pelos desenvolvedores para definir as maneiras pelas quais os componentes são combinados, customizados e utilizados. O Prospero é implementado em CLOS (*Common Lisp Object System*) e é direcionado para encapsular as complexidades técnicas de baixo nível. O Prospero não oferece suporte à construção da interface com o usuário, pressupondo a utilização de outro toolkit com este propósito.

O COPSE-Web [David & Borges, 2004] é uma extensão do ambiente COPSE (*Collaborative Project Support Environment*) [Dias & Borges, 1999], voltado para a construção de groupware para web. O COPSE-Web adota uma arquitetura baseada no estilo MVC (Modelo-Visão-Controle). A infra-estrutura da plataforma oferece vários serviços de execução às ferramentas colaborativas e um

framework de classes a partir do qual as ferramentas são instanciadas. A infra-estrutura provê às ferramentas recursos de gerenciamento de projetos, processos, documentos, percepção e perfil. Algumas das ferramentas já disponíveis no ambiente são: quadro de avisos, relatório de eventos, fórum de discussão e agenda, oferecendo suporte a pré-reunião, reunião e pós-reunião.

Anderson et al. [2002] propõem uma abordagem para construção de toolkits denominada Dragonfly, que mantém um link bidirecional entre a arquitetura conceitual e a implementação, possibilitando ao desenvolvedor uma transição suave entre os níveis. A abordagem foi utilizada na construção do toolkit TeleComputing Developer (TCD).

Guicking et al. [2005] propõem o framework Agilo, voltado para integração de aplicações colaborativas. O framework flexibiliza a infra-estrutura de comunicação e os modelos de distribuição, de compartilhamento, de concorrência e de sincronização, oferecendo para eles implementações recorrentes.

2.4. Engenharia do Domínio e Componentes

A engenharia de domínio objetiva disponibilizar componentes que implementam os conceitos de um domínio de software em particular e possam ser reusados para implementar novas aplicações deste domínio [Werner & Braga, 2005]. Ao mapear conceitos do domínio, aumenta-se a chance de reuso em todas as fases do desenvolvimento, desde a análise até a implantação, e não é considerada somente uma única aplicação, mas sim uma família. Esta abordagem torna o grau de reuso de um dado componente maior, assim como seu entendimento, uma vez que será diretamente mapeado no domínio da aplicação (menor distância semântica) [D'Souza & Wills, 1998, p.720]. O componente é codificado (espaço da solução) de acordo com as necessidades do domínio (espaço do problema). A engenharia do domínio possibilita a uniformidade dos conceitos tratados pelos envolvidos no projeto e representados nas diversas etapas de desenvolvimento e produtos advindos do processo.

A engenharia do domínio engloba a análise do domínio, o projeto de uma arquitetura orientada ao domínio e a implementação dos componentes

correspondentes [Werner & Braga, 2005]. A análise do domínio é o processo de identificar e organizar o conhecimento sobre uma classe de problemas para apoiar sua descrição e solução [Pietro-Diaz & Arango, 1991]. Na analise do domínio é identificado, colecionado, organizado e representado o conhecimento advindo dos sistemas, da teoria de apoio, da tecnologia e do desenvolvimento de um domínio de interesse [Peterson, 1991].

A análise do domínio serve de guia para o processo de construção e reuso de componentes [Pietro-Diaz & Arango, 1991], sendo que ao reusar um elemento em um nível, o elemento correspondente no nível mais abstrato também é reusado [Werner & Braga, 2005]. Na análise do domínio, nenhuma solução de software é assumida. O propósito da modelagem é entender os conceitos e seus relacionamentos, sendo o principal produto desta atividade a definição de um modelo do domínio, que estrutura os conceitos, as regras, o vocabulário, o contexto, as funcionalidades e os relacionamentos presentes no domínio. A análise do domínio guia a forma de pensar, no que diz respeito à predição, explicação ou derivação de fatos sobre o domínio, e fornece uma classificação e organização das características dos sistemas daquele domínio [Arango, 1994].

Uma vez modelado o domínio, são desenvolvidas diversas aplicações distintas com base no mesmo modelo. A modelagem embasa a criação de ferramentas, técnicas e componentes para oferecer suporte às diversas atividades do desenvolvimento de software, instrumentando todo o desenvolvimento. A análise do domínio é feita consultando especialistas no domínio e a literatura da área ou com base no conhecimento adquirido pelos desenvolvedores durante o processo de desenvolvimento ou de utilização de diversas aplicações de um mesmo domínio. Outra fonte bastante utilizada para a análise do domínio é o estudo de aplicações desenvolvidas, no intuito de levantar as funcionalidades recorrentes, invariantes e opcionais [Werner & Braga, 2005]. A análise do domínio sistematicamente extrai características dos sistemas de uma mesma família [Griss, 2001, p.410]. O modelo fornece um vocabulário compartilhado entre os desenvolvedores e usuários, o que facilita a comunicação e o reuso, e oferece uma noção da abrangência do domínio e de suas relações [Werner & Braga, 2005].

Há vários processos de engenharia de domínio na literatura, dentre eles, o FODA (*Feature Oriented Domain Analysis*) [Kang et al., 1990], o FORM (*Feature Oriented Reuse Method*) [Kang et al., 1998] e o RSEB (*Reuse-Driven Software Engineering Business*) [Jacobson et al., 1997]. No contexto desta tese, é utilizado o processo CBD-Arch-DE [Blois et al., 2004], que é uma evolução do processo Odyssey-ED [Braga, 2000], por ser mais direcionado e instrumentado para o desenvolvimento baseado em componentes. O processo CDB-Arch-DE organiza a engenharia do domínio em 4 atividades principais: planejamento do domínio, análise do domínio, projeto do domínio e implementação do domínio, conforme ilustrado na Figura 2.13.

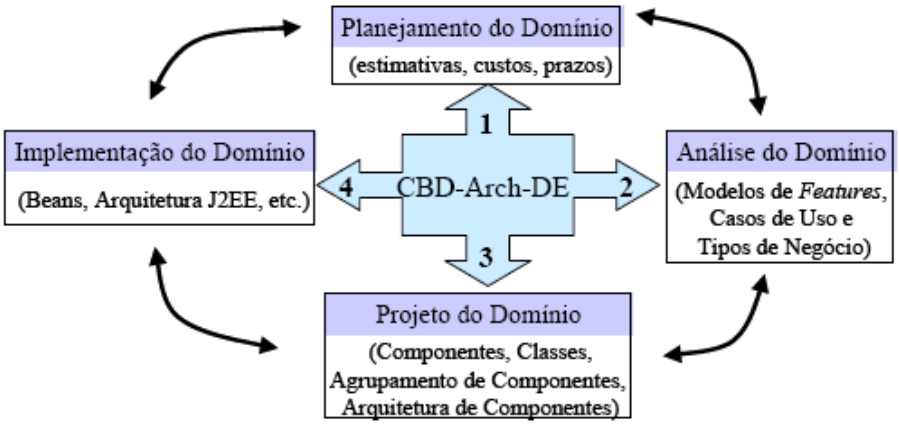

Figura 2.13. Atividades do processo CBD-Arch-DE [Blois et al., 2004]

A Figura 2.14 ilustra as atividades da análise do domínio no CBD-Arch-DE. A análise do domínio se inicia pela identificação dos contextos que representam os sub-domínios do domínio principal. Após esta atividade, são identificadas as *features* (características) correspondentes, criando um modelo de características. As características são obtidas de especialistas, usuários, documentações e de aplicações já existentes e modelam os aspectos variáveis e invariáveis do domínio, em um alto nível de abstração. As características são classificadas em conceituais, funcionais e tecnológicas e definidas como obrigatórias ou opcionais. Estas propriedades são mapeadas para os demais artefatos do domínio, que são gerados com base no modelo de características. Após a identificação das características, são identificados os tipos de negócio, que representam os aspectos estáticos do domínio candidatos a persistência no contexto do projeto, e os casos de uso associados.

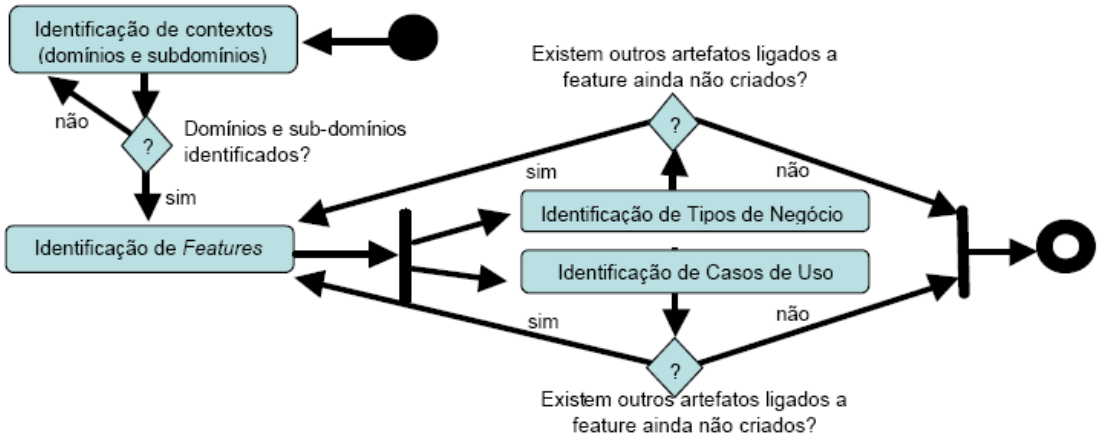

Figura 2.14. Análise do domínio no processo CBD-Arch-DE [Blois et al., 2004]

O projeto do domínio envolve a criação, composição e geração de uma arquitetura de componentes e de suas interfaces. A criação de componentes do domínio é efetuada com base em estilos arquiteturais baseado em tipos de negócio, sendo especificados o conjunto de classes que os implementam, seus métodos, atributos e relacionamentos. Após a definição dos componentes e interfaces, procura-se agrupar os componentes relacionados, por critérios de acoplamento e coesão, para obter um grau de granularidade adequado. A última atividade do processo é a implementação do domínio. Nesta etapa, é feito o mapeamento dos componentes do domínio para uma tecnologia de componentes específica.

O uso de um processo de engenharia do domínio guia a especificação dos componentes e os torna mais propícios para o reuso, por prever o domínio e os componentes nas diversas etapas do desenvolvimento do sistema. Nesta tese, o processo de engenharia do domínio é utilizado para definir o conjunto de componentes do *component kit* utilizado.

2.5. Considerações Finais

Na literatura são encontradas diversas abordagens para instrumentar o desenvolvimento de groupware. Neste capítulo foram apresentados exemplos de requisitos de groupware [Tietze, 2001; Schmidt & Rodden, 1996; Mandviwalla & Olfman, 1994], UML estendida [Rubart & Dawabi, 2002], padrões específicos [Groupware Patterns Swiki, 2005; Lukosch & Schümmer, 2004; Santoro et al.,

2001], arquiteturas de groupware [Tietze, 2001; Dewan, 1998], frameworks [Prakash & Knister, 1994; Lee et al., 2002; Kirsch-Pinheiro et al., 2002; Nunamaker et al., 2001; Buzko et al., 2000], e técnicas de avaliação de groupware [Baker et al., 2001; Araujo et al., 2004]. A engenharia de domínio, o desenvolvimento baseado em componentes e o modelo 3C de colaboração são utilizados para conectar este ferramental e torná-lo interoperável.

O desenvolvimento baseado em componentes tem se mostrado uma abordagem de desenvolvimento bastante promissora para o desenvolvimento de groupware [Blois & Becker, 2002]. Na literatura, há diversas propostas de utilização de componentes de software na construção de groupware [Banavar et al., 1998; Marsic, 1999; Won et al., 2005; Litiu & Prakash, 2000; Roth & Unger, 2000; Koch & Koch, 2000; Grundy et al., 1997; Slagter & Biemans, 2000; Chabert et al., 1998; Li & Muntz, 1998; Roseman & Greenberg, 1996; Hummes & Merialdo, 2000; Begole et al., 1999; Guerrero & Fuller, 1999; Siqueira et al., 2003; Isenhour et al., 1997]. Entretanto, nenhuma delas utiliza o modelo 3C de colaboração e uma engenharia de domínio como base para a concepção e organização dos componentes e do processo de desenvolvimento.

Szyperski [2003] afirma que há quatro motivações principais para utilizar componentes de software. A primeira e mais antiga é relacionada com a idéia de *mercado de componentes*. Nesta visão, as empresas buscam e adquirem componentes necessários para resolver problemas específicos e integram estes componentes ao sistema sendo desenvolvido. A segunda motivação é relacionada com *linha de produto*. São desenvolvidos componentes com o objetivo de reusá-los em diversos sistemas, reduzindo o custo total de investimento e os custos de manutenção. A terceira está relacionada com a idéia de composição pelo usuário final (*tailorability*). Disponibiliza-se uma infra-estrutura onde o usuário implanta componentes (*deployment*) visando estender a capacidade do sistema. A quarta motivação está relacionada com a utilização de *serviços dinâmicos*, descobertos e instalados na medida da necessidade. Nesta abordagem, ao necessitar de uma determinada funcionalidade, o sistema consulta catálogos de serviços e instala e se re-configura. A utilização desta abordagem possibilita a construção de sistemas que evoluem para acompanhar novas demandas. Web services é uma tecnologia que vem sendo utilizada com este propósito [Hansen et al., 2005].

Plataforma	Objetivo	Motivação [Szyperski, 2003]	Modelo de Componentes	Linguagem
Live [Banavar et al., 1998]	Groupware síncrono	linha de produto	Extensão do JavaBeans	Java
DISCIPLINE [Marsic, 1999]	Groupware síncrono para educação	linha de produto	Extensão do JavaBeans	Java
FreEvolve [Won et al., 2005]	Groupware distribuído	tailorability	Extensão do JavaBeans (FlexiBeans)	Java
DACIA [Litiu & Prakash, 2000]	Groupware para dispositivos móveis	linha de produto	Proprietário	Java
DreamTeam [Roth & Unger, 2000]	Groupware síncrono	linha de produto	Proprietário	Java
IRIS [Koch & Koch, 2000]	Edição colaborativa de documentos multimídia	linha de produto	Proprietário	Java
JViews [Grundy et al., 1997]	Sistemas colaborativos com múltiplas visões	linha de produto	Proprietário	Java
CoCoWare [Slagter & Biemans, 2000]	Groupware em geral	tailorability	Extensão do .Net	.Net
Habanero [Chabert et al., 1998]	Groupware síncrono	linha de produto	Proprietário	Java
COCA [Li & Muntz, 1998]	Flexibilidade na coordenação	tailorability	Proprietário	Java
GroupKit [Roseman & Greenberg, 1996]	Groupware síncrono	linha de produto	Proprietário	Tcl/Tk
Lumis (www.lumis.com.br)	Portais web	tailorability	Proprietário	Asp.NET
Mambo (www.mamboserver.com)	Portais web	tailorability	Proprietário	PHP
XOOPS (www.xoops.org)	Portais web	tailorability	Proprietário	PHP
Zope (www.zope.org)	Portais web	tailorability	Proprietário	Python
DotNetNuke (www.dotnetnuke.com)	Portais web	tailorability	Proprietário	Asp.NET / VB.NET
ACOST [Hummes & Merialdo, 2000]	Ferramentas de comunicação síncrona	tailorability	Extensão do JavaBeans	Java
Flexible JAMM [Begole et al., 1999]	Applets colaborativos	linha de produto	Swing	Java
TOP [Guerrero & Fuller, 1999]	Aplicações colaborativas na Web	linha de produto	Proprietário	Java
MoCA [Sacramento et al., 2004]	Colaboração em dispositivos móveis	linha de produto	Proprietário	Java
Sieve [Isenhour et al., 1997]	Visualização colaborativa de dados	linha de produto	Extensão do JavaBeans	Java
Prospero [Dourish, 1998]	Groupware em geral	linha de produto	Proprietário	CLOS
COPSE-Web [David & Borges, 2004]	Groupware para Web	linha de produto	Proprietário	Java
TeleComputing Developer [Anderson et al. , 2002]	Groupware em geral	linha de produto	Proprietário	Java
Agilo [Guicking et al., 2005]	Groupware síncrono	linha de produto	Proprietário	Java

Tabela 2.1. Plataformas para o desenvolvimento de groupware baseado em componentes

As abordagens encontradas na literatura com relação à utilização de componentes de software no desenvolvimento de groupware se concentram basicamente na segunda e terceira motivação identificadas por Szyperski [2003] (linha de produto e *tailorability*), conforme ilustrado na Tabela 2.1. Os sistemas oferecem um ferramental para construir aplicações colaborativas a partir de componentes interoperáveis e, alguns deles, oferecem a possibilidade de composição pelos usuários-finais. Também pode ser notado na Tabela 2.1 que a maioria das plataformas analisadas utiliza a linguagem Java como base para sua implementação e não há um modelo de componentes padrão.

O foco da abordagem proposta nesta tese é na segunda motivação (linha de produto), visando instrumentar o desenvolvedor de groupware na montagem de sistemas colaborativos. Entretanto, alguma flexibilidade é oferecida ao usuário final, que seleciona e instala novos serviços de colaboração. Isto os possibilita, até certo ponto, adaptar a aplicação colaborativa para suas necessidades específicas e acompanhar as características do grupo e das tarefas.

A engenharia do domínio é propícia para utilização em domínios que apresentam processos e características complexos, onde há dificuldade de modelagem utilizando os processos tradicionais [Braga, 2000], que é o caso de CSCW e groupware. O modelo 3C instrumenta a análise do domínio e o desenvolvimento de groupware como um todo. O modelo 3C é tratado em mais detalhes no próximo capítulo.