Uma comparação técnica das abordagens existentes para implementação de requisitos colaborativos em aplicações não colaborativas

Mauro C. Pichiliani[1], Celso M. Hirata[1]

[1]Divisão de Ciência da Computação – Instituto Tecnológico de Aeronáutica (ITA)
Caixa Postal 12.228-900 - São José dos Campos - SP

{pichilia, hirata}@ita.br

Abstract. *Many approaches can be used to facilitate the implementation of collaborative requirements on non collaborative applications. Each approach uses different design techniques and has specific objectives and pre-conditions. During the implementation of collaborative requirements is not always easy to decide which approach is the most recommended i.e. which criteria should be used to compare the approaches. Based on the literature work, this paper presents a set of technical criteria to technically compare the existing approaches to implement collaborative requirements on non collaborative applications. With these criteria, the developers can choose the most suitable approach to implement collaborative requirements on non collaborative applications.*

Resumo. *Existem diversas técnicas para facilitar a implementação de requisitos colaborativos em aplicações não colaborativas. Cada abordagem usa técnicas de projeto diferentes e possui pré-requisitos e objetivos específicos. Na implementação dos requisitos colaborativos, nem sempre é imediato decidir qual é a abordagem mais recomendada i.e. quais devem ser os critérios comparativos. A partir do estudo bibliográfico, este trabalho apresenta um conjunto de critérios para comparar tecnicamente as abordagens existentes. Deste modo, espera-se que a comparação técnica e os critérios apresentados neste trabalho auxiliem os desenvolvedores na implementação de requisitos colaborativos em aplicações não colaborativas.*

1. Introdução

A implementação de requisitos colaborativos em aplicações não colaborativas possui o potencial de aumentar significativamente a disponibilidade de aplicações colaborativas e melhorar a usabilidade dessas aplicações [Xia et al. 2004]. Durante a última década, vários esforços na área de CSCW (*Computer Supported Collaborative Work*) contribuíram para a redução da complexidade necessária para estender e adaptar aplicações não colaborativas, com o objetivo de apoiar a colaboração.

Atualmente, existe uma variedade de abordagens de extensão e adaptação de aplicações não colaborativas, contudo, não se dispõe de critérios de comparação das abordagens disponíveis para implementar requisitos colaborativos em aplicações não colaborativas. Isto se deve, em parte, à heterogeneidade das abordagens, que variam desde a utilização de aplicações externas para emular eventos até a substituição de componentes da aplicação.

Os principais requisitos colaborativos implementados em aplicações colaborativas são: (i) comunicação, utilizada para trocar idéias, discutir, aprender, negociar ou para tomar decisões; (ii) coordenação, que organiza o grupo para evitar que esforços de comunicação e cooperação sejam perdidos e que as tarefas sejam realizadas na ordem e no tempo correto; (iii) percepção (*awareness*), que faz com que um participante fisicamente separado esteja ciente da presença e das ações dos demais participantes do grupo; e (iv) compartilhamento de informações, encarregado de prevenir esforços repetitivos e assegurar que todos os participantes do grupo estejam utilizando a mesma informação.

Critérios técnicos de comparação de abordagens podem auxiliar os desenvolvedores na escolha da abordagem a ser utilizada, além de fornecer uma contextualização do uso de cada uma das abordagens. Deste modo, o desenvolvedor pode compreender melhor qual tipo de abordagem é mais recomendada na implementação dos requisitos colaborativos desejados, de acordo com seu contexto.

O objetivo deste artigo é apresentar um conjunto de critérios para as abordagens de implementação de requisitos colaborativos e uma comparação técnica das abordagens utilizando os critérios de comparação. A partir da comparação de abordagens apresentada neste trabalho, os desenvolvedores podem analisar a conveniência de uma determinada abordagem. Tento em vista que a implementação de requisitos colaborativos apresenta vários desafios, possuir uma comparação técnica de abordagens facilita o processo de desenvolvimento.

Este artigo está dividido em quatro seções. A Seção 2 descreve as abordagens utilizadas para a implementação de requisitos colaborativos em aplicações existentes. Na Seção 3 apresentam-se os critérios comparativos e a comparação técnica propriamente dita. Por fim, na Seção 4 são apresentadas as conclusões e alguns comentários finais.

2. Abordagens Existentes

Esta seção descreve as abordagens existentes na literatura para construir aplicações colaborativas e para transformar aplicações não colaborativas existentes em aplicações colaborativas. As abordagens descritas nesta seção incluem o uso de *toolkits*, sistemas de colaboração transparente, substituição dos componentes da aplicação por componentes colaborativos, adaptação transparente e mapeamento de componentes.

A estratégia tradicional para a modificação de uma aplicação, isto é, a modificação da aplicação de forma *ad hoc*, não será abordada nesta comparação, pois esta abordagem proporciona pouca ou nenhuma técnica sistemática de desenvolvimento que possa ser seguida para transformar aplicações não colaborativas existentes em aplicações colaborativas.

2.1 *Toolkits*

Numa tentativa de reduzir a complexidade de implementação de sistemas *groupware*, vários pesquisadores têm explorado o uso de *toolkits*. Segundo Ellis et al. [Ellis et al. 1991], sistemas *groupware* são sistemas baseados em computadores que apóiam grupos de pessoas engajadas em uma tarefa ou objetivo comum e que provém uma interface para um ambiente compartilhado.

Um *toolkit* é definido como um conjunto de componentes predefinidos e reutilizáveis, com a finalidade de oferecer ferramentas e infra-estrutura suficientes para permitir que o programador desenvolva sistemas *groupware* de alta qualidade com razoável esforço [Greenberg & Roseman 1998].

Os *toolkits* para a construção de sistemas *groupware* facilitam a criação de novas aplicações colaborativas por meio de componentes de grupo e controles de percepção construídos para serem reutilizáveis, permitindo a criação de poderosas aplicações colaborativas. Da mesma forma que casas pré-fabricadas podem ser construídas a partir da montagem de módulos pré-fabricados, o uso de *toolkits* na construção de software colaborativo pode tornar a tarefa do desenvolvedor mais simples e, muitas vezes, mais rápida, uma vez que o trabalho principal consiste apenas na montagem e configuração dos componentes em vez da criação deles a partir do zero.

Devido a estas características, pode-se dizer que o uso de *toolkits* para a construção de sistemas *groupware* segue, de uma maneira geral, os princípios envolvidos no Desenvolvimento Baseado em Componentes.

O Desenvolvimento Baseado em Componentes (DBC) têm como objetivo prover o reuso dos componentes no nível de implementação e aumentar a interoperabilidade entre as partes do software. Os princípios envolvidos no Desenvolvimento Baseado em Componentes demandam características específicas dos componentes, como a capacidade de prover funcionalidades específicas, a baixa dependência entre componentes e a suscetibilidade a modificações.

Toolkits amplamente utilizados, como o GroupKit [Gutwin & Greenberg 2002], o COAST (*COoperative Application Systems Toolkit*) [Schuckmann et al. 1996] e o MAUI (*Multi-User Awareness UI toolkit*) [Hill & Gutwin 2004], fornecem *frameworks*, ambientes, bibliotecas de funções, componentes e controles para tornar mais rápida e fácil a construção de novas aplicações colaborativas.

O reuso dos componentes de *toolkits* pode ajudar o desenvolvimento de novos softwares colaborativos, entretanto, os componentes fornecidos pelo *toolkit* não podem auxiliar a adaptação de uma aplicação não colaborativa existente. Em geral isso acontece devido à necessidade de implementação de detalhes específicos do *toolkit*, que nem sempre são compatíveis com a estrutura da maioria das aplicações existentes.

Um exemplo desta incompatibilidade é a diferença entre a estrutura de uma aplicação existente a estrutura do *toolkit*. Além disso, detalhes técnicos como a linguagem de programação e a arquitetura utilizada na construção dos componentes do *toolkit* podem tornar o reuso dos componentes inviável em certas situações. Mesmo nos casos onde os detalhes técnicos não inviabilizam a utilização do *toolkit*, a quantidade de esforço necessário para a reutilização dos componentes pode não compensar.

Apesar dos *toolkits* não serem muito úteis para ajudar a extensão de um software existente, eles podem fornecer inspiração e idéias necessárias para implementar aspectos essenciais de comunicação, colaboração e cooperação em softwares existentes.

2.2 Sistemas de Colaboração Transparente

Os Sistemas de Colaboração Transparente (SCT) fornecem um meio de compartilhar o uso de aplicações existentes apenas de forma síncrona, onde as interações dos participantes são simultâneas ou separadas por pequenos períodos de tempo. Estes

sistemas permitem apenas a forma síncrona de interação entre os participantes, pois eles emulam as entradas e saídas de dados dos usuários da aplicação, fornecendo aos usuários a impressão de que todos utilizam a mesma interface da aplicação, ao mesmo tempo e no mesmo local. De acordo com Begole et al. [Begole et al. 1999], o termo transparente é utilizado por estes sistemas porque o compartilhamento fornecido é transparente ou desconhecido para a aplicação e seus desenvolvedores. O compartilhamento, neste contexto, envolve as aplicações legadas com interação monousuária.

Estes sistemas utilizam o conceito de caixa preta de colaboração transparente, onde alguma aplicação externa é responsável pelas comunicações e notificações necessárias para apoiar aspectos básicos da colaboração entre os usuários remotos. Os mecanismos por trás desses sistemas incluem uma arquitetura centralizada, a captação e entrega dos dados de entrada dos usuários para múltiplos locais e a transmissão da interface gráfica da aplicação para todos os participantes. Exemplos de SCT comerciais incluem o Microsoft Netmeeting [Microsoft 2007], o XTV [Abdel-Wahab & Feit 1991] e o SharedX [Garnkel et al. 1994].

Colaboração Transparente Inteligente, ou ICT (*Intelligent Collaboration Transparency*) [Li & Li 2002], apresenta uma evolução dos SCT, pois eles utilizam uma infra-estrutura compartilhada entreposta entre aplicações heterogêneas e o sistema operacional para compartilhar os ambientes gráficos de cada local.

Devido à heterogeneidade das aplicações envolvidas, a simples reprodução dos eventos de uma aplicação em outras aplicações diferentes não faz sentido. Uma operação semântica de uma aplicação geralmente é implementada de forma diferente em outra aplicação, requerendo um mecanismo que, além de compreender as operações dos usuários, deve traduzir essas operações em seqüência de operações equivalentes a outras aplicações, antes que elas sejam reproduzidas. Deste modo, o mecanismo deve ser inteligente o suficiente para traduzir as operações de uma aplicação em uma seqüência de operações compatíveis em outras aplicações, por meio do conhecimento semântico das ações realizadas pelos usuários.

Com o uso deste tipo de sistemas, os usuários podem colaborar em uma tarefa comum, através de suas aplicações favoritas. Contudo, um módulo de captura/reprodução de eventos específico para cada plataforma deve ser utilizado para tratar da comunicação e do controle de concorrência, propiciando a interoperabilidade entre as aplicações.

Uma segunda geração de aplicações baseadas na abordagem do ICT foi elaborada por Lu et al. [Lu et al 2004], denominada ICT2. A principal diferença entre a abordagem ICT clássica e a ICT2 é que a última não intercepta e interpreta os eventos da aplicação. Em vez disso, a ICT2 utiliza uma versão adaptada de um algoritmo que gera as seqüências de edição entre os diferentes estados dos documentos locais dos usuários. Para organizar as ações concorrentes dos usuários, uma versão modificada do algoritmo de transformação operacional é utilizada.

Mangan [Mangan 2006] propõe uma variação da técnica de colaboração transparente. Esta proposta chama-se Arquitetura de Colaboração Transparente (ACT) e amplia as arquiteturas transparentes com o acréscimo do conceito de eventos semânticos. Neste contexto, os eventos descrevem o significado das operações realizadas pelo usuário da aplicação e são gerados a partir de interpretações dos eventos

de entrada e dos eventos de aplicação das arquiteturas transparentes. Com estes eventos, é possível alimentar os modelos de percepção que oferecem informações úteis para mecanismos: de apoio à colaboração assíncrona; de descoberta de perfil de usuário; e de histórico das experiências dos usuários.

Apesar de apresentar uma alternativa rápida para compartilhar aplicações sem modificações no código fonte, os SCTs da primeira geração, a ACT e as abordagens ICT e ITC2 receberam muitas críticas no sentido que elas não suportam a colaboração adequadamente. O motivo principal dessas críticas envolve o uso ineficiente dos recursos de rede e a imposição de um estilo de colaboração inflexível e altamente acoplado que não suporta algumas funcionalidades comuns aos sistemas *groupware* como, por exemplo, controle de concorrência, interface WYSIWIS (*What You See Is What I See*) relaxada, percepção em grupo e tarefas colaborativas delegadas [Begole et al. 1999]. Uma interface WYSIWIS relaxada, em particular, propicia aos usuários o controle sobre aspectos de suas interface individuais permitindo que eles trabalhem de forma mais natural, sendo considerada uma funcionalidade determinando para a ação de um *groupware*.

2.3 Adaptação Transparente

A abordagem chamada Adaptação Transparente foi proposta para ajudar a implementar a colaboração em aplicações comerciais, quando os desenvolvedores não têm acesso ao código fonte da aplicação. Esta abordagem é baseada no compartilhamento da aplicação e no uso de uma Interface de Programa de Aplicação (API) para interceptar as interações locais do usuário, convertê-las em operações abstratas, manipular estas operações por técnicas colaborativas e reproduzir as operações modificadas, por meio da API, nos locais remotos de colaboração [Xia et al. 2004].

O termo transparente é utilizado porque esta abordagem não requer nenhuma mudança no código fonte da aplicação. Contudo, esta abordagem requer que o fabricante da aplicação forneça uma API capaz de gerenciar os dados da aplicação e implementar os mecanismos de colaboração.

Diferentemente dos ambientes de compartilhamento de aplicações, os quais não requerem nenhum tratamento específico, a abordagem de Adaptação Transparente requer uma nova camada de software para implementar a arquitetura replicada que fornecerá o compartilhamento de eventos e dados da aplicação. De acordo com Xia et al. [Xia et al. 2004], a combinação da arquitetura replicada e da abordagem de Adaptação Transparente permite vários benefícios como, por exemplo, uma grande quantidade de interações entre os usuários, um controle de concorrência eficiente e uma melhor interface WYSIWIS relaxada.

Desta maneira, o código fonte da aplicação não é necessário. Contudo, uma API completa e que forneça o acesso aos eventos e dados gerados pela aplicação deve ser fornecida. Mesmo com uma API completa, alguns métodos específicos de controle de concorrência e dispositivos de presença são impossíveis de implementar na abordagem de Adaptação Transparente, devido à necessidade da mudança no código fonte da aplicação para implementar alguns requisitos colaborativos.

2.4 Substituição de Componentes

A abordagem de substituição de alguns componentes de uma aplicação por componentes colaborativos foi introduzida pelo Flexible JAMM (*Java Applets Made Multi-user*) [Begole et al. 1999], sendo conhecida também como Sistema de Colaboração Transparente Flexível. Na comparação apresentada neste trabalho, esta abordagem é denominada Substituição de Componentes, pois esta descrição identifica rapidamente a técnica utilizada na implementação da abordagem.

Esta abordagem permite que os usuários trabalhem juntos de forma colaborativa ou independente, por meio da substituição dinâmica de certos objetos da interface do usuário por componentes de interfaces colaborativas [Begole et al. 1999]. Para utilizar esses componentes colaborativos, a aplicação que recebe os componentes deve possuir certas características como, por exemplo, a capacidade de migração de processos, recursos para a substituição de componentes em tempo de execução e a habilidade para interceptar e reproduzir os eventos gerados pelo usuário.

Os componentes da interface gráfica de usuário que o Flexible JAMM fornece incluem mecanismos de percepção como barras de rolagem multiusuário, visão radar e *telepointers*. Estes componentes propiciam, respectivamente, nomes de usuários em diferentes barras de rolagem, visão em miniatura da área de trabalho comum e ponteiros virtuais representando usuários remotos.

A principal limitação desta abordagem reside no fato que os componentes oferecidos pelo Flexible JAMM suportam apenas o requisito colaborativo percepção, não apresentando componentes para os requisitos colaborativos comunicação, coordenação e compartilhamento de informações. Além disso, somente um pequeno grupo de aplicações existentes possui todos os requisitos necessários para o uso dos componentes do Flexible JAMM, ou seja, apenas as aplicações que atendam aos requisitos de migração de processos, substituição de componentes em tempo de execução e a habilidade para interceptar e reproduzir os eventos podem utilizar os componentes oferecidos pelo Flexible JAMM. Estas características limitam o uso do Flexible JAMM a poucas aplicações, evitando que esta abordagem possa ser utilizada de forma sistemática em softwares de diferentes domínios de conhecimento.

2.5 Mapeamento de Componentes

O Mapeamento de Componentes, proposto por Pichiliani & Hirata [Pichiliani & Hirata 2006], sugere um mapeamento dos principais componentes de aplicações não colaborativas, que devem estar baseadas no estilo arquitetural MVC (*Model-View-Controller*), para componentes de uma aplicação colaborativa com o objetivo de permitir a colaboração síncrona entre participantes. Usando o mapeamento, aplicações existentes podem ser estendidas para apoiar a colaboração síncrona durante a elaboração de tarefas compartilhadas.

De acordo com os autores, o Mapeamento de Componentes permite um tratamento uniforme de um conjunto de requisitos colaborativos mínimos, em um nível alto de abstração, com o objetivo de fornecer um ponto de partida para os desenvolvedores de aplicações, sob o ponto de vista conceitual.

Esta abordagem requer que a aplicação não colaborativa, donde é feito o Mapeamento de Componentes, estruture seus componentes de acordo com o estilo

arquitetural MVC, que sugere a separação dos componentes que manipulam dados da aplicação (o modelo), dos componentes de tratamento de dados (o controlador) e também dos componentes responsáveis interface de visualização (a visão).

Além de requerer a organização dos componentes da aplicação não colaborativa no estilo arquitetural MVC, o Mapeamento de Componentes implica na modificação do código fonte da aplicação para alterar o comportamento dos componentes, de acordo com os requisitos de colaboração especificados. A partir do uso do mapeamento, a aplicação não colaborativa passa a suportar requisitos colaborativos, compartilhando os dados através de uma arquitetura híbrida, onde um Servidor de Colaboração é responsável pela a comunicação dos dados manipulados pelos componentes do Modelo.

A principal desvantagem do Mapeamento de Componentes é a necessidade de modificação direta do código fonte da aplicação. Esta necessidade envolve diversos recursos que nem sempre são disponíveis, como o código fonte da aplicação ou um compilador compatível com a linguagem de programação da aplicação. Deste modo, o uso desta abordagem se torna inviável em algumas ocasiões. Contudo, pelo fato de se trabalhar no código fonte, esta abordagem oferece uma grande flexibilidade na implementação dos requisitos colaborativos.

3. Comparação Técnica de Abordagens

Esta seção descreve os critérios técnicos utilizados para comparar as abordagens apresentadas na seção anterior. Em seguida, a comparação entre as abordagens que transformam aplicações não colaborativas existentes em aplicações colaborativas é apresentada.

3.1 Critérios de Comparação

Uma vez que as abordagens existentes foram apresentadas, é necessário comparar suas características por meio de critérios técnicos. Diversos critérios podem ser escolhidos para a comparação, porém os critérios escolhidos nesta comparação entre abordagens visam auxiliar o desenvolvedor, concentrando-se nas características e nos detalhes envolvidos no uso das abordagens, obtidas por meio dos trabalhos relacionados e pelo estudo experimental das abordagens, conduzido pelos autores deste trabalho. Os critérios de comparação são os seguintes: Código Fonte, Requisito Tecnológico, Propósito, Arquitetura e Flexibilidade.

Um critério a ser considerado antes da adoção de alguma abordagem leva em consideração a disponibilidade do código fonte. Este critério está diretamente relacionado com o modelo de licença que a aplicação segue. Os modelos de licença que as aplicações seguem, para efeitos de comparação de abordagens, podem ser classificados, basicamente, em dois tipos: aplicações abertas, isto é, aplicações que possuem uma licença onde o código fonte é disponibilizado publicamente e que permite a modificação sem nenhum ônus; e aplicações proprietárias, onde a licença não permite que o código fonte seja modificado. Na comparação de abordagens apresentada nesta seção, quatro valores auto-explicativos são especificados para o critério Código Fonte, a saber: ‘Requer o código fonte (da aplicação)’; ‘Não requer o código fonte (da aplicação)’; ‘Requer API do sistema operacional’; e ‘Requer API da aplicação’.

Outro critério importante a ser considerado na comparação das abordagens envolve os requisitos tecnológicos relacionados com a implementação. Estes requisitos

influenciam diretamente o uso da aplicação a partir do momento que a abordagem for implementada. Por exemplo, a abordagem de *toolkits* requer uma linguagem de programação específica para que o *toolkit* possa ser utilizado, o que pode causar problemas de portabilidade e integração decorrentes da necessidade deste requisito. O critério Requisito Tecnológico, utilizado na comparação das abordagens realizada nesta seção, pode assumir os seguintes valores: 'Nenhum', 'Depende da linguagem de programação', 'Requer camada de software', 'Depende da linguagem do componente' e 'Requer aplicação no estilo MVC'.

É fundamental compreender o propósito das abordagens antes de considerar sua utilização. Enquanto algumas abordagens demandam a modificação direta do código fonte, outras devem ser consideradas apenas para novas aplicações. No critério Propósito, a abordagem pode possuir o valor "Criar novas aplicações", para indicar o contexto de construção de novas aplicações por meio da abordagem, ou o valor "Promover colaboração", para indicar que a abordagem é utilizada para promover a colaboração em uma aplicação já existente. Nota-se que uma abordagem pode conter os dois valores para este critério, como é o caso da abordagem 'Substituição de Componentes'.

A arquitetura da aplicação colaborativa obtida a partir da utilização de qualquer uma das abordagens apresentadas na Seção 2 é um critério a ser considerado nesta comparação de abordagens. Suthers [Suthers 2001] apresenta algumas maneiras pelas quais as aplicações colaborativas de ensino organizam seus componentes na arquitetura MVC, de acordo com o nível de acoplamento dos componentes, isto é, de acordo com o nível de dependência entre os componentes. Apesar de focar apenas em aplicações colaborativas na área de ensino, as arquiteturas apresentadas por Suthers são genéricas a ponto de poderem ser consideradas para aplicações colaborativas de propósito gerais.

A Figura 1 apresenta os quatro tipos de organização do modelo MVC sugeridos por Suthers: 1) Arquitetura Centralizada; 2) Arquitetura Replicada; 3) Arquitetura Distribuída; e 4) Arquitetura Híbrida. Apesar de não ser exaustiva, as arquiteturas apresentadas por Suthers representam como a maioria dos sistemas, *groupware* ou não, são organizados. Caracterizar a arquitetura da abordagem é importante, pois a arquitetura e os casos de uso orientam o projeto detalhado e a implementação de aplicações no processo de desenvolvimento orientado a objetos.

Deste modo, o critério Arquitetura indica qual das arquiteturas pode ser utilizada a partir da escolha da abordagem, assumindo os valores 'Centralizada', 'Replicada', 'Distribuída' e 'Híbrida'. De forma análoga ao critério Propósito, uma mesma abordagem pode usar diferentes arquiteturas.

O último critério técnico utilizado na comparação envolve a Flexibilidade das abordagens para implementar requisitos colaborativos. Este critério indica a capacidade que as abordagens detêm para satisfazer requisitos. Este critério pode assumir os valores subjetivos 'Pouca', 'Média', e 'Muita', para indicar que a abordagem apresenta, respectivamente, pouca, média e muita flexibilidade na implementação de requisitos colaborativos. Este critério é superficial, cujo objetivo é apenas indicar de forma geral os diferentes níveis de flexibilidade que cada abordagem proporciona.

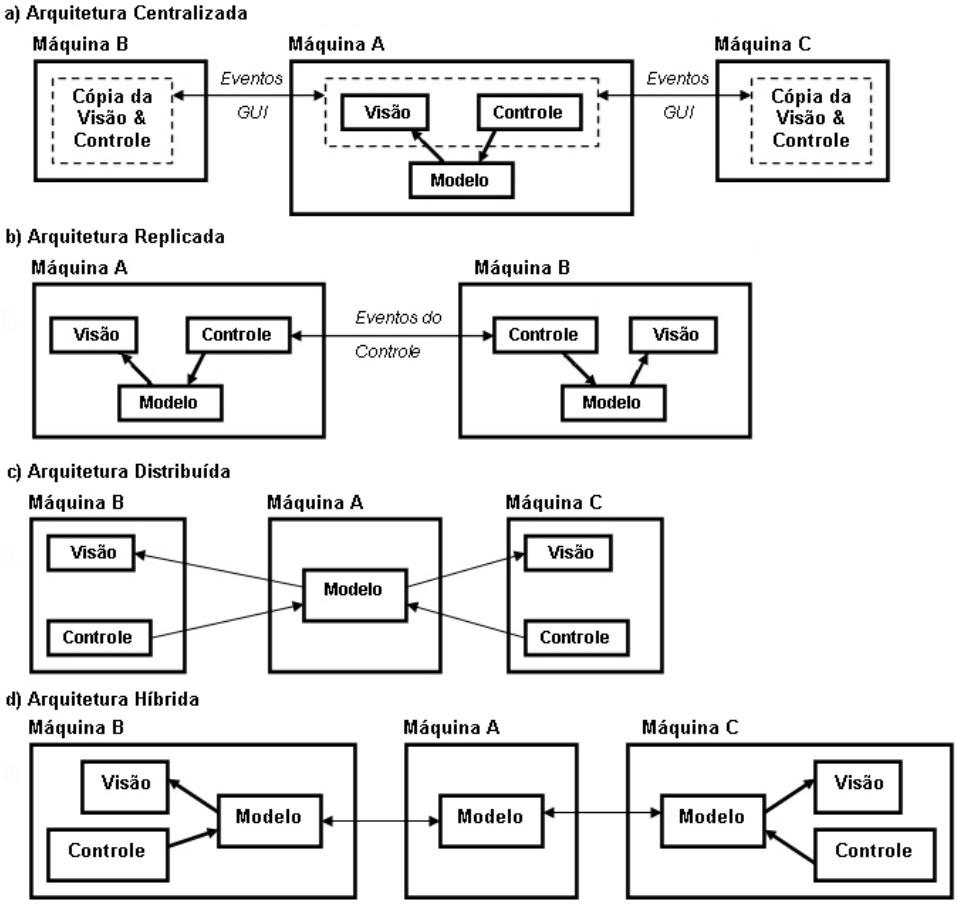

Figura 1. Organizações dos componentes do MVC em: a) Arquitetura Centralizada, b) Arquitetura Replicada, c) Arquitetura Distribuída e d) Arquitetura Híbrida. Adaptada de [Suthers 2001].

As seguintes abordagens, apresentadas na Seção 2, são comparadas neste trabalho: a abordagem *toolkit*, a primeira geração de Sistemas de Colaboração Transparente (SCT), as abordagens ICT, ICT2 e ACT da segunda geração de SCT, a abordagem de Adaptação Transparente, a Substituição de Componentes e o Mapeamento de Componentes. A Tabela 1 apresenta a comparação de abordagens de acordo com os critérios estabelecidos.

3.2 Discussão da Comparação Técnica de Abordagens

A abordagem '*Toolkits*' é utilizada para a construção de novas aplicações colaborativas e não permite a promoção de colaboração entre aplicações já existentes. Devido a esta característica, esta abordagem não se aplica no caso de implementação de requisitos colaborativos a partir de uma aplicação não colaborativa. Contudo, para as situações onde uma nova aplicação colaborativa deve ser construída, a abordagem '*Toolkits*' deve ser considerada. Como esta abordagem faz uso de componentes prontos e da arquitetura Centralizada, o desenvolvedor pode criar rapidamente novas aplicações colaborativas.

Por ser a primeira abordagem apresentada para o desenvolvimento de aplicações colaborativas, os *toolkits* se tornaram populares, sendo utilizados para a construção de diversos protótipos e provas de conceito.

A abordagem 'SCT - 1º geração' permite o uso colaborativo de, virtualmente, qualquer aplicação, pois esta abordagem encasula as aplicações existentes agindo como uma plataforma operacional. Como nenhuma modificação no código fonte é sugerida por esta abordagem, o desenvolvedor deve apenas configurar a plataforma operacional

para que ela suporte o uso colaborativo da aplicação. Apesar de permitir o uso colaborativo de aplicações, Begole et al. [Begole et al. 1999] criticam esta abordagem afirmando que ela não suporta a colaboração adequadamente, argumentando que esta abordagem utiliza os recursos de rede de forma ineficiente, impondo um estilo de colaboração inflexível e altamente acoplado.

Tabela 1. Comparações de abordagens.

Abordagem	Código Fonte	Requisito Tecnológico	Propósito	Arquitetura	Flexibilidade
Toolkits	Não requer o código fonte	Depende da linguagem de programação	Criar novas aplicações	Centralizada	Muita
SCT - 1º geração	Não requer o código fonte	Nenhum	Promover colaboração	Centralizada	Pouca
SCT - 2º geração (ICT, ICT2 e ACT)	Requer API do sistema operacional	Requer camada de software	Promover colaboração	Centralizada ou Distribuída	Pouca
Adaptação Transparente	Requer API da aplicação	Requer camada de software	Promover colaboração	Replicada	Média
Substituição de Componentes	Requer o código fonte da aplicação	Depende da linguagem do componente	Criar novas aplicações e Promover colaboração	Replicada	Pouca
Mapeamento de Componentes	Requer o código fonte dos componentes da aplicação	Requer aplicação no estilo MVC	Promover colaboração	Híbrida	Muita

A abordagem 'SCT - 2º geração' também não sugere modificações no código fonte da aplicação. Contudo, para utilizar esta abordagem é necessário manipular a API do sistema operacional, que geralmente é complexa e extensa. Além do conhecimento necessário para manipular a API do sistema operacional, o desenvolvedor também deve conhecer as principais operações que o usuário pode realizar na aplicação, além dos detalhes específicos do cada sistema operacional que a aplicação for executada.

Devido a esta característica, o desenvolvedor se distancia do foco principal, que é tornar a aplicação colaborativa, ao se aprofundar na programação da API do sistema operacional. Além disso, o compartilhamento fornecido por esta abordagem se limita à replicação de eventos, não fornecendo recursos para o desenvolvedor implementar os demais aspectos funcionais comuns a aplicações colaborativas. Em contrapartida, esta abordagem é a única que permite a colaboração entre aplicações heterogêneas.

A próxima abordagem listada na Tabela 1, a abordagem 'Adaptação Transparente', possui como requisito tecnológico uma API da aplicação. Este tipo de requisito limita o uso desta abordagem a aplicações bem documentadas e que possuam uma API que permita a interceptação e reprodução de eventos, apresentado as mesmas

limitações da segunda geração de sistemas de colaboração transparente. O desenvolvedor que utilizar esta abordagem deve conhecer profundamente tanto à aplicação como sua API, uma vez que é por meio desta última que todas as funcionalidades colaborativas são implementadas.

Em comparação com as abordagens '*Toolkits*', 'SCT - 1º geração', 'SCT - 2º geração', 'Substituição de Componentes' e 'Mapeamento de Componentes', a principal vantagem da abordagem 'Adaptação Transparente' é a separação da construção de um *groupware* da construção de uma camada de software, que propicia as funcionalidades colaborativas. Portanto, a abordagem 'Adaptação Transparente' implica no desenvolvimento de uma camada de software para capturar e replicar os eventos gerados pelas aplicações.

Com base na experiência do autor no uso da abordagem 'Adaptação Transparente', a alteração direta do código fonte da aplicação pode requerer menos recursos do que a utilização da API da aplicação. A aplicação da abordagem 'Adaptação Transparente' apresenta bons resultados, especialmente em aplicações comerciais complexas, como o editor de texto Word [Xia et al. 2004].

As abordagens 'Substituição de Componentes' e 'Mapeamento de Componentes' são as únicas abordagens que sugerem uma modificação direta na aplicação. Desta maneira, o desenvolvedor tem mais liberdade para implementar os aspectos colaborativos da maneira que desejar. Contudo, a manipulação direta do código fonte da aplicação pode acarretar em diversos problemas como, por exemplo, a inserção de novos defeitos na aplicação.

A abordagem 'Substituição de Componentes' sugere a troca de determinados componentes da aplicação por componentes colaborativos. Seguindo esta abordagem, o desenvolvedor substitui apenas partes específicas da aplicação para torná-la colaborativa. Além de demandar a modificação direta da aplicação, uma vez que um componente existente da aplicação será substituído por outro, esta abordagem é limitada pela oferta de componentes compatíveis com a tecnologia utilizada pela aplicação. Outra característica desta abordagem, descrita na Subseção 2.4, indica que apenas os requisitos relacionados à percepção podem ser implementados por esta abordagem.

A abordagem 'Substituição de Componentes' é abordagem que possui muitos pontos em comum com a abordagem 'Mapeamento de Componentes', em relação aos critérios utilizados nesta comparação. Contudo, o Mapeamento de Componentes se destaca em relação à Substituição de Componentes por não requerer recursos técnicos específicos, como uma determinada linguagem de programação ou ambiente de desenvolvimento, durante a implementação da abordagem. É interessante notar que, em certas aplicações, as abordagens 'Substituição de Componentes' e 'Mapeamento de Componente' podem ser utilizadas em conjunto, especialmente quando o mapeamento indicar que algum componente não pode ser modificado e deve ser substituído. Contudo, para que o desenvolvedor utilize estas duas abordagens em conjunto, é necessário que todos os requisitos de ambas as abordagens sejam satisfeitos.

De acordo com os dados da Tabela 1, o Mapeamento de Componentes deve ser utilizado no contexto das aplicações que possuem código fonte disponível, ou ao menos o código fonte dos principais componentes que constituem a aplicação. Devido a este valor para este critério, a maioria das aplicações candidatas para o uso do Mapeamento são as aplicações de código livre, onde um grupo de desenvolvedores disponibiliza o

código fonte para livre acesso. Também é importante notar que, em comparação com as outras abordagens, o Mapeamento de Componentes é o único que requer não apenas a estruturação da aplicação em componentes, mas também uma organização lógica dos mesmos, pois o estilo arquitetural MVC é um dos requisitos do mapeamento.

A abordagem 'Mapeamento de Componentes' apresenta um requisito adicional implícito no uso da arquitetura híbrida: a necessidade de um Servidor de Colaboração. Este Servidor de Colaboração deve ser desenvolvido a partir dos componentes do Modelo, o que implica em mais esforço de desenvolvimento.

Comparando o critério Disponibilidade do Código Fonte das abordagens 'SCT - 1º geração' 'SCT - 2º geração', 'Adaptação Transparente', 'Substituição de Componentes' e 'Mapeamento de Componentes', pode-se notar que as abordagens se dividem em dois grupos: o grupo de abordagens que não requer nenhum tipo de desenvolvimento, caracterizado pelo uso de plataformas operacionais, e o grupo das abordagens que requerem o desenvolvimento de infra-estrutura para a utilização colaborativa das aplicações, caracterizado pela necessidade de desenvolvimento.

O critério Requisito Tecnológico indica que as abordagens 'SCT - 2º geração', 'Adaptação Transparente', 'Substituição de Componentes' e 'Mapeamento de Componentes' requerem uma API do sistema operacional, uma API da aplicação, o código fonte da aplicação e o código fonte dos componentes da aplicação, respectivamente, além de recursos adicionais, com compilador, ferramentas de desenvolvimento etc. Estes requisitos nem sempre estão disponíveis ao desenvolvedor, limitando o uso destas abordagens a um conjunto específico de aplicações.

Com relação às arquiteturas utilizadas pelas abordagens da Tabela 1, Suthers [Suthers 2001] indica que a arquitetura Centralizada utiliza ineficientemente os recursos de rede, pois esta arquitetura transmite a informação completa de visualização e eventos da interface pela rede. Ainda segundo Suthers, a arquitetura Replicada possui a desvantagem de ser naturalmente mais adequada para aplicações WYSIWIS restritas e não WYSIWIS relaxadas. Já a arquitetura Distribuída é criticada por depender muito da rede, tornando-se inutilizável quando os serviços de rede estão indisponíveis. Suthers também critica a arquitetura Híbrida, argumentando que sua implementação é mais complexa do que as demais.

Sob o ponto de vista da Flexibilidade, as abordagens 'SCT - 1º geração' e 'SCT - 2º geração' apresentam pouca capacidade para fornecer requisitos colaborativos, pois elas não modificam a aplicação. De forma análoga, a abordagem 'Substituição de Componentes' limita-se apenas ao requisito percepção, pois esta abordagem sugere a substituição apenas dos componentes da interface gráfica.

A abordagem 'Adaptação Transparente' recebeu o valor 'Média' para o critério Flexibilidade, pois apesar de não modificar a aplicação, pode-se contar com os recursos da API para estender algumas partes da aplicação e implementar alguns requisitos colaborativos. As abordagens 'Mapeamento de Componentes' e ' *Toolkits*' possuem muita flexibilidade para implementar novos requisitos, devido à sua estratégia de criar novas aplicações a partir do zero ou modificar diretamente o código fonte.

Todas as abordagens comparadas, com exceção dos *toolkits*, permitem que aplicações não colaborativas possam apresentar recursos colaborativos, possibilitando a reutilização de aplicações existentes em um contexto colaborativo. Esta contribuição é

relevante, pois possibilita associar em uma mesma solução as ferramentas já existentes, adequadas à tarefa e provavelmente familiares aos participantes, com as funcionalidades colaborativas necessárias para realizar a tarefa em ambientes distribuídos colaborativos.

4. Conclusões e Comentários Finais

Este artigo apresentou os critérios e uma comparação de abordagens disponíveis para implementar requisitos colaborativos em aplicações não colaborativas. A partir da comparação de abordagens apresentada neste trabalho, os desenvolvedores podem compreender melhor qual tipo de abordagem é mais recomendada, em termos dos critérios apresentados, na implementação dos requisitos colaborativos desejados, auxiliando os desenvolvedores na escolha da abordagem a ser utilizada.

As abordagens '*Toolkits*', 'SCT - 1º geração', 'SCT - 2º geração (ICT, ICT2 e ACT)', 'Adaptação Transparente', 'Substituição de Componentes', e 'Mapeamento de Componentes' foram apresentadas de acordo com o contexto na qual elas devem ser utilizadas. Os critérios 'Código Fonte', 'Requisito Tecnológico', 'Propósito', 'Arquitetura' e 'Flexibilidade' foram utilizados para a comparação, de acordo com o estudo bibliográfico e a experiência prática dos autores no uso das abordagens.

Para uma tomada de decisão efetiva, além dos critérios técnicos, devem-se considerar também os critérios gerenciais. Um critério gerencial importante que deve ser incluído na escolha final da abordagem é o esforço de análise, projeto, implementação e testes, se possível em termos quantitativos, como estimativas de Homens-Hora, ou eventualmente qualitativos. Esse tipo de critério gerencial tem valores específicos resultantes de um conjunto de atributos e valores da organização de desenvolvimento tais como: prontidão e habilidades dos membros da equipe, técnicas, ferramentas e processos adotados. Apesar dos critérios gerenciais serem importantes na decisão de qual abordagem adotar, a elaboração desses critérios gerencial foge ao escopo deste artigo. De qualquer forma, os autores julgam que critérios gerenciais na prática devam ser considerados na decisão final sobre escolha da abordagem.

Com a comparação apresentada neste trabalho, os desenvolvedores de aplicações colaborativas podem contar com um apoio para a escolha de uma determinada abordagem. Tento em vista que a implementação de requisitos colaborativos apresenta vários desafios, contar com critérios de comparação pode facilitar o processo de adoção de uma determinada abordagem, incentivando a comunidade de desenvolvedores a modificar aplicações existentes para suportar funcionalidades colaborativas e, conseqüentemente, aumentando a quantidade de aplicações colaborativas nos diversos domínios de conhecimento.

Agradecimentos. Os autores gostariam de agradecer aos revisores anônimos que contribuíram para a melhoria deste trabalho.

5. Referências

Abdel-Wahab, H., Feit, M. A. (1991) "XTV: A Framework for Sharing X Window Clients in Remote Synchronous Collaboration", em: Proceedings of the IEEE Tricomm, Chapel Hill, Carolina do Norte, E.U.A., p.159-167.

Begole, J. C. A., Rosson, M. B., Shaffer, C. A. (1999) "Flexible collaboration transparency: supporting worker independence in replicated application sharing

systems", em: ACM Transactions on Computer-Human Interaction, volume 6, número 2, Junho de 1999, p.95-132.

Ellis, C., Gibbs, S. J, Rein, G. L. (1991) "Groupware: some issues and experiences", em: Communications of the ACM, volume 34, número 1, Janeiro de 1991, p.38-58.

Garnkel, D., Welti B. C., Yip, T. W. (1994) "SharedX: A tool for real-time collaboration", em: HP Journal, volume 45, número 2, Abril de 1994, p.23-36.

Greenberg, S., Roseman, M (1998) Groupware toolkits for synchronous work, John Wiley & Sons, Nova York, primeira edição.

Gutwin, C., Greenberg, S. (2002) "A Descriptive Framework of Workspace Awareness for Real-Time Groupware", em: Computer-Supported Cooperative Work, volume 11, número 3, Julho de 2002, p.411- 446.

Hill, J., Gutwin, C. (2004) "The MAUI Toolkit: Groupware Widgets for Group Awareness", em: Computer Supported Cooperative Work, volume 13, números 5-6, Dezembro de 2004, p.539-571.

Li, D., Li, R. (2002) "Transparent sharing: interoperation of heterogeneous single-user applications", em: Proceedings of the 8th ACM Conference on Computer Supported Cooperative Work (CSCW'02), New Orleans, Lousiana, E.U.A., p.246-255.

Lu, J., Li, R., Li, D. (2004) "A State Difference Based Approach to Sharing Semi-Heterogeneous Single-User Editors", em: Proceedings of the Sixth International Workshop on Collaborative Editing Systems (IWCES6) in CSCW'04, Chicago, Illinois, E.U.A., 2004.

Mangan, M. A. S. (2006) Uma abordagem para o desenvolvimento de apoio à percepção em ambientes colaborativos de software Tese (Doutorado em Engenharia de Sistemas e Computação) - Universidade Federal do Rio de Janeiro, Rio de Janeiro.

Microsoft. "NetMeeting Home", web site acessado em abril/2007: http://www.microsoft.com/windows/netmeeting/.

Pichiliani, M. C., Hirata, C. M. (2006) "A Guide to map application components to support multi-user real-time collaboration", em: Proceedings of the 2nd International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCon 2006), Georgia, E.U.A., 2006.

Schuckmann, C., Kirchner, L., Schümmer, J., Haake, J. M. (1996) "Designing object-oriented synchronous groupware with COAST", em: Proceedings of the 3rd ACM Conference on Computer Supported Cooperative Work (CSCW'96), Nova York, E.U.A., p.30-38.

Suthers D. (2001) "Architectures for Computer Supported Collaborative Learning", em: Proceedings of the IEEE International Conference on Advanced Learning Technologies (ICALT 2001), New Orleans, Lousiana, E.U.A., p.6-8, 2001.

Xia S., Sun D., Sun, C., Chen D., Shen H. (2004) "Leveraging Single-user Applications for Multi-user Collaboration: the CoWord Approach", em: Proceedings of the 9th ACM Conference on Computer Supported Cooperative Work (CSCW'04), Chicago, Illinois, E.U.A., p.162-171.