Uma Arquitetura para Resolução de Conflitos Coletivos em Sistemas Ubíquos e Cientes de Contexto

Thais R. M. B. Silva[1,2], **Linnyer B. Ruiz**[3], **Antonio A. F. Loureiro**[1]

[1] Departamento de Ciência da Computação – Universidade Federal de Minas Gerais (UFMG)
31.270-010 – Belo Horizonte – MG – Brasil

[2]Universidade Federal de Viçosa (UFV) – Campus de Florestal
35.690-000 – Florestal – MG – Brasil

[3]Departamento de Informática – Universidade Estadual de Maringá (UEM)
87.020-900 – Maringá – PR – Brasil

thaisrb@dcc.ufmg.br, linnyer@gmail.com, loureiro@dcc.ufmg.br

Resumo. *Sistemas ubíquos executam aplicações computacionais onipresentes, capazes de prover seus serviços o tempo todo e em todo lugar. Em geral, esses sistemas são coletivos uma vez que são projetados para operar em ambientes cotidianos compartilhados por grupos de pessoas. Aplicações ubíquas utilizam frequentemente informações sobre seus usuários e o ambiente de execução, as quais são chamadas de contexto, para adaptar seus serviços. Nesse caso, essas aplicações podem alcançar um estado de incerteza quando os dados contextuais de cada usuário considerado apontam para diferentes ações de adaptação. Esse trabalho propõe uma arquitetura cliente-servidor para detecção e resolução de conflitos em sistemas coletivos, ubíquos e cientes de contexto, cujos módulos consideram o compromisso entre qualidade de serviços e consumo de recursos. Duas aplicações coletivas e cientes de contexto diferentes foram avaliadas. Os resultados mostram que a arquitetura proposta foi capaz de manter a qualidade dos serviços e estender o tempo de vida das aplicações.*

Abstract. *Ubiquitous systems perform omnipresent computational applications that provide their services all the time and everywhere. In general, these systems are collective as they are designed to operate into everyday environments shared by a group of people. Ubiquitous applications frequently use information about their users and the execution environment, which are called contexts, to adapt their services. In this case, these applications may reach an uncertain state when the contextual data from each considered user points to different adaptation actions. This work proposes a client-server architecture to detect and solve conflicts for collective, ubiquitous and context-aware systems, whose modules consider the trade-off between quality of services and resources consumption. Two different collective context-aware applications were evaluated. The results showed that the proposed architecture was able to maintain the quality of the services and extend the application lifetime.*

1. Introdução

Aplicações computacionais cientes de contexto podem ser definidas como aquelas que utilizam informações sobre entidades (objetos, pessoas ou ambientes) de interesse para

adaptarem seus serviços com o objetivo de aumentar a satisfação dos usuários [Dey 2001]. A informação considerada por esse tipo de aplicação é chamada de contexto e pode incluir dados do ambiente (e.g., temperatura e indicadores de chuva) e características pessoais (e.g., sentimentos e estado físico). A sensibilidade ao contexto possui uma estreita relação com a computação ubíqua, a qual define sistemas com capacidades de computação e comunicação sem fio integradas ao ambiente físico, utilizadas de maneira transparente e sob-demanda [Weiser 1993]. Sistemas ubíquos normalmente são executados por dispositivos que possuem restrições de recursos, tais como energia, memória e processamento. O uso de contextos em aplicações ubíquas ajuda a melhorar a personalização e automatização dos serviços, além de contribuir para a utilização eficiente dos recursos disponíveis, prolongando o tempo de vida das mesmas [Huebscher et al. 2006].

Apesar de já existirem diversos trabalhos na literatura abordando questões sobre sistemas ubíquos e cientes de contexto, muitos aspectos ainda precisam ser resolvidos para que os mesmos possam ser construídos e utilizados em larga escala na prática. Uma dessas questões está relacionada à característica de coletividade de aplicações desse tipo, ou seja, o compartilhamento das mesmas por dois ou mais usuários. Em geral, apesar dos usuários nesses cenários possuírem objetivos comuns, eles podem divergir sobre as adaptações desejadas para os serviços oferecidos, devido às diferenças em seus perfis individuais e/ou à escassez de recursos. Dessa forma, conflitos podem ser detectados durante a adaptação de serviços, e devem ser resolvidos considerando os interesses do grupo, assim como os individuais. Conflitos de interesses ocorrem frequentemente em aplicações ubíquas e cientes de contexto, uma vez que as mesmas são, em geral, projetadas para operar em ambientes cotidianos, os quais são normalmente compartilhados por diversos usuários. Nesse caso, além da eficiência em qualidade de serviços (QoS) individual e coletiva, o serviço de resolução de conflitos deve ser flexível, robusto e consumir poucos recursos.

Este trabalho propõe uma arquitetura para o tratamento de incompatibilidades entre os perfis dos usuários e/ou com o ambiente compartilhado. Até onde os autores puderam pesquisar, não existem trabalhos na literatura que tratem desses conflitos considerando sistematicamente o compromisso entre QoS e consumo de recursos, conforme necessário para as aplicações ubíquas. Além disso, o tema é em geral pouco abordado na literatura. As poucas soluções encontradas propõem a utilização de um único algoritmo sem considerar as características atuais dos usuários e seus dispositivos, da aplicação e do ambiente compartilhado. Essa abordagem muitas vezes não é interessante, principalmente em sistemas ubíquos, visto que as características citadas possuem impacto direto no tipo de tratamento mais interessante a ser realizado em caso de conflitos, e seus valores podem sofrer modificações ao longo do tempo.

A arquitetura proposta está baseada em um arcabouço definido para que aplicações cientes de contexto coletivas detectem e resolvam conflitos, e em uma metodologia elaborada como uma implementação particular dos módulos do arcabouço, para aplicações relacionadas à computação ubíqua. Para avaliar o comportamento e o desempenho da arquitetura proposta, bem como da metodologia particular definida para sistemas ubíquos e cientes de contexto, dois diferentes estudos de caso foram simulados. As implementações realizadas podem ser vistas como provas de conceito para demonstrar o funcionamento e a flexibilidade da arquitetura para resolução de conflitos proposta.

O restante deste trabalho está organizado da seguinte maneira: a seção 2 discute algumas soluções para resolução de conflitos em aplicações cientes de contexto coletivas encontradas em trabalhos relacionados disponíveis na literatura. A seção 3 apresenta os principais conceitos relacionados ao tema de pesquisa. A seção 4 descreve a arquitetura proposta para modelar o tratamento de conflitos de interesse. A seção 5 apresenta as definições e os resultados de ambos os estudos de caso desenvolvidos. Finalmente, a seção 6 lista os comentários finais e alguns direcionamentos para trabalhos futuros.

2. Trabalhos Relacionados

[Roy et al. 2006] apresentam uma solução para o desenvolvimento de uma casa inteligente que considera as atividades e a localização de múltiplos habitantes. Sua principal contribuição é prover um ambiente que se adapta a cada participante sem conceder preferência a qualquer um deles. Esse trabalho é restrito a uma aplicação específica e às suas características relacionadas.

[Shin and Woo 2005] discutem o problema da adaptação para ambientes ubíquos colaborativos. Os autores propõem a atribuição de prioridades para cada usuário e sempre aplicam a mesma solução estática para resolver os problemas de conflito. [Shin et al. 2008] também propuseram um esquema de resolução que pode ser automático ou manual, dependendo do tipo de contexto em conflito. [Park et al. 2005] desenvolveram um esquema baseado em preferências, que diminui a quantidade de relutância de todos os usuários envolvidos em um conflito. Todos esses trabalhos apresentam propostas que utilizam um único algoritmo para resolver conflitos, independente das condições atuais da aplicação, dos dispositivos dos usuários, da rede sem fio, dos perfis individuais, dentre outros aspectos.

[Capra et al. 2003] apresentam uma arquitetura de middleware para aplicações móveis e cientes de contexto chamada CARISMA. Uma vez que utiliza políticas para adaptar serviços cientes de contexto, as aplicações podem demandar adaptações conflitantes. Nesse caso, CARISMA propõe um algoritmo de leilão para selecionar a política a ser aplicada. Essa estratégia é baseada em princípios de micro-economia. Apesar de simples, dinâmico e adaptável, o esquema é ainda baseado em um único algoritmo com possibilidades limitadas.

3. Conceitos Básicos

Essa seção apresenta alguns conceitos necessários para o melhor entendimento do presente trabalho.

Definição 1 (Aplicação Coletiva) *É uma aplicação ciente de contexto que possui um grupo de usuários interessados em executar suas tarefas, as quais deverão ser adaptadas de acordo com os interesses e dados contextuais dos membros desse grupo.*

Uma aplicação coletiva é composta por um conjunto de tarefas (individuais e coletivas) e um conjunto de usuários. As tarefas são os serviços providos pela aplicação, sendo coletivas aquelas executadas simultaneamente por dois ou mais usuários. Exemplos desse tipo de aplicação são guias turísticos coletivos e ambientes inteligentes compartilhados.

Definição 2 (Contexto Coletivo) *Representa uma informação relevante para uma aplicação coletiva, que será utilizada para caracterizar as condições do ambiente, assim como interesses ou preferências de cada participante.*

Definição 3 (Conflito Coletivo) *Pode ser definido como um estado inconsistente alcançado por uma aplicação coletiva após avaliar contextos coletivos ambientais e pessoais. A aplicação se torna incapaz de realizar adaptações de maneira a satisfazer interesses individuais e coletivos ao mesmo tempo.*

Definição 4 (Resolução de Conflito Coletivo) *Adoção de um algoritmo ou técnica para resolver conflitos coletivos identificados durante a execução de uma aplicação coletiva.*

4. Uma Arquitetura para Detectar e Resolver Conflitos Coletivos

A seção 2 descreve trabalhos encontrados na literatura que apresentam soluções específicas para determinados tipos de aplicações, considerando um conjunto pré-determinado de contextos, usuários e ambientes. O presente trabalho possui como objetivo propor uma solução mais completa, abrangente e flexível para a resolução de conflitos coletivos em aplicações ubíquas e cientes de contexto.

Essa seção apresenta uma arquitetura para resolução de conflitos coletivos, cujo modelo é centralizado com rodízio de servidores. Os módulos para detecção e resolução de conflitos, executados pelo servidor, foram genericamente definidos em um arcabouço chamado *Conflict Engine*. Além disso, foi proposta uma instância particular desse arcabouço para sistemas coletivos, ubíquos e cientes de contexto capaz de considerar o compromisso entre consumo de recursos e QoS. Os critérios de QoS para as aplicações consideradas neste trabalho são a satisfação individual e coletiva dos usuários com os resultados obtidos pelas resoluções dos conflitos.

4.1. Modelo de Arquitetura

Neste trabalho, uma arquitetura centralizada para execução das atividades de detecção e resolução de conflitos coletivos foi proposta. Dessa maneira, um dos participantes da aplicação deverá assumir o papel de servidor, recebendo e agregando todos os dados contextuais necessários, realizando o processamento relacionado e finalmente notificando as ações realizadas sobre as tarefas da aplicação aos demais membros do grupo. Caso os usuários das aplicações executem seus processos em dispositivos ubíquos, os quais possuem importantes restrições de recursos, essa abordagem poderia levar ao desgaste da fonte de energia do participante escolhido como servidor, impossibilitando-o de continuar ativo até o término da aplicação. Para solucionar o problema em questão, um esquema de rodízio para o papel de servidor foi definido. Nesse esquema, todo membro do grupo assume as atividades de servidor durante um período de tempo da execução da aplicação. Foi considerado que todos os dispositivos utilizados pelos usuários possuem as mesmas características, ou seja, são homogêneos.

Em linhas gerais, o esquema de rodízio implementado para o modelo de arquitetura funciona da seguinte maneira:

- Inicialmente, um dos participantes é escolhido aleatoriamente para ser o servidor. Todos os demais participantes são notificados da decisão.

- Para cada tarefa da aplicação a ser executada, o servidor executa as atividades de detecção e resolução de conflitos.
- O participante servidor atual indica um outro participante para assumir o papel de servidor. Será escolhido aleatoriamente um participante que ainda não tenha desempenhado as atividades de servidor. Caso todos já tenham assumido tal papel, inicia-se novamente o mesmo esquema de rodízio. Todos os demais participantes são notificados da decisão.

O modelo de arquitetura proposto foi definido de maneira a funcionar dentro da própria rede de comunicação estabelecida entre os dispositivos da aplicação, e independente de elementos externos a ela, tais como servidores instalados em outras máquinas. A organização centralizada foi escolhida devido à sua simplicidade e consequentemente menor custo computacional. Finalmente, o esquema de rodízio definido permite uniformizar o consumo de recursos entre os participantes da aplicação. A implementação de um mecanismo simplificado atende à necessidade de repasse das atividades de servidor entre os participantes, sem consumir uma quantidade significativa de recursos dos dispositivos. As atividades do servidor são sempre executadas por um dispositivo ubíquo, pertencente a um dos usuários envolvidos na aplicação.

4.2. O Arcabouço *Conflict Engine*

Esse trabalho apresenta um arcabouço chamado *Conflict Engine* que define módulos responsáveis pela detecção e resolução de conflitos, a serem executados pelo servidor da arquitetura para tratamento de conflitos coletivos. O arcabouço proposto é genérico e pode ser implementado de acordo com as características de cada aplicação coletiva.

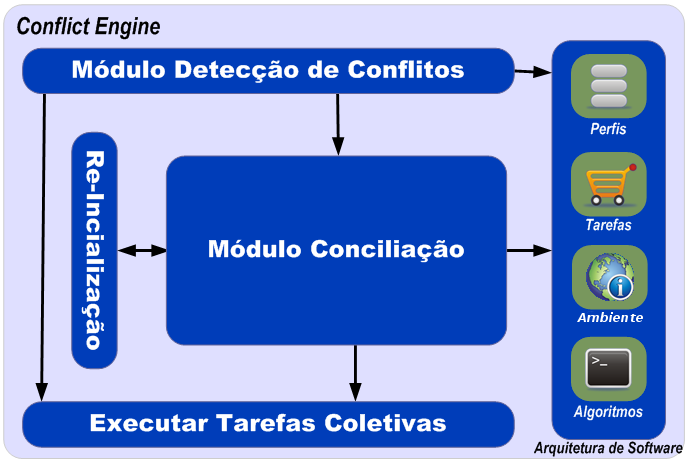

Figura 1. Módulos Componentes - *Conflict Engine*

A figura 1 mostra a relação existente entre a *Conflict Engine* e a arquitetura de software ciente de contexto utilizada pela aplicação ciente de contexto coletiva. A arquitetura de software é a responsável por organizar e executar os serviços de aquisição, processamento e armazenamento de dados contextuais [Baldauf et al. 2007], os quais são utilizados pelos módulos propostos no arcabouço. Por outro lado, a arquitetura utiliza o serviço de processamento de tarefas coletivas provido pela *Conflict Engine* e, de acordo com a necessidade das aplicações, armazena as informações por ela produzidas. Os módulos que compõem a *Conflict Engine* são descritos a seguir e também estão ilustrados na figura 1.

O *Módulo Detecção de Conflitos* é responsável por identificar impasses de adaptação das tarefas da aplicação entre os participantes e notificá-los ao *Módulo Conciliação* sempre que necessário. Ele executa uma análise tri-dimensional considerando os perfis dos usuários envolvidos, os contextos dos ambientes compartilhados e as tarefas da aplicação. Essa análise verificará a compatibilidade entre as três dimensões consideradas, identificando se existe ou não a possibilidade de conciliar as demandas individuais e coletivas, dados os recursos disponíveis no ambiente, para cada tarefa da aplicação. Quando conflitos coletivos não forem detectados, as tarefas analisadas são liberadas para execução. Caso contrário, o *Módulo de Detecção de Conflitos* notifica ao *Módulo Conciliação* quais são as tarefas para as quais foram detectados conflitos, bem como as possíveis causas para os mesmos.

O objetivo do *Módulo Conciliação* é adaptar as tarefas da aplicação de acordo com os interesses individuais e coletivos, considerando ainda os recursos disponíveis no ambiente, sempre que conflitos coletivos forem detectados. O módulo utiliza os perfis dos usuários relacionados, os dados contextuais do ambiente, e quaisquer outras informações necessárias armazenadas nas bases de dados da arquitetura de software ciente de contexto utilizada pela aplicação. Para resolver conflitos, o módulo executa um algoritmo ou técnica de resolução, escolhida de acordo com as características da aplicação ciente de contexto coletiva considerada. Como parte da análise para verificação de conflitos ou durante o ajuste das tarefas, pode ser necessário executar ações do *Módulo de Reinicialização*, as quais são responsáveis pela troca de dados entre os dispositivos participantes da aplicação coletiva.

Vale ressaltar que, os módulos propostos para o arcabouço *Conflict Engine* definidos acima, serão executados pelo participante escolhido como servidor durante a execução de cada tarefa, para detectar e resolver os conflitos coletivos relacionados. O arcabouço entretanto é genérico, e não define a maneira como cada módulo será implementado. A seguir, este trabalho apresenta uma implementação para os módulos da *Conflict Engine*, considerando que os mesmos serão executados por dispositivos de aplicações coletivas, ubíquas e cientes de contexto.

4.3. Uma Metodologia Flexível para Detectar e Resolver Conflitos

Comportamentos específicos foram definidos para os módulos da *Conflict Engine*, os quais configuram uma nova metodologia para resolver conflitos em aplicações coletivas, ubíquas e cientes de contexto. A metodologia propõe implementações flexíveis para os módulos de detecção e resolução de conflitos, isto é, implementações únicas para cada um desses módulos, que são parametrizadas para serem utilizadas por diferentes aplicações.

No *Módulo Detecção de Conflitos*, uma análise sobre as três dimensões consideradas como entradas para o mesmo (tarefas da aplicação, perfis dos usuários e características do ambiente) é realizada. Cada aplicação indica ao módulo quais dessas dimensões devem ser consideradas, bem como quais características ou parâmetros das mesmas devem ser especificamente analisadas. A combinação formada entre essas informações providas por cada aplicação indica para a metodologia quais são as circunstâncias que podem levar o sistema a um estado de inconsistência. Uma análise é realizada pelo módulo sobre essa combinação, resultando na detecção ou não de conflitos coletivos para a aplicação corrente.

No caso em que conflitos são de fato detectados, essa situação é repassada ao *Módulo Conciliação*. A característica chave da implementação definida pela metodologia para esse módulo é a seleção dinâmica de algoritmos para resolução de conflitos coletivos. Nesse caso, o objetivo é escolher, a cada momento, o algoritmo que atenda da melhor maneira possível às demandas por satisfação individual e coletiva das aplicações, realizando um consumo consciente dos recursos disponíveis para as mesmas. A metodologia possui um repositório de algoritmos contendo opções diversificadas de implementações para resolver conflitos relacionados a diferentes aplicações. Esse repositório de algoritmos é armazenado junto à arquitetura de software ciente de contexto utilizada. Um repositório de meta-dados relacionados aos algoritmos também foi definido, indicando parâmetros tais como complexidade, consumo médio de energia e indicadores de QoS médios. Esses dados são utilizados durante a escolha dos algoritmos.

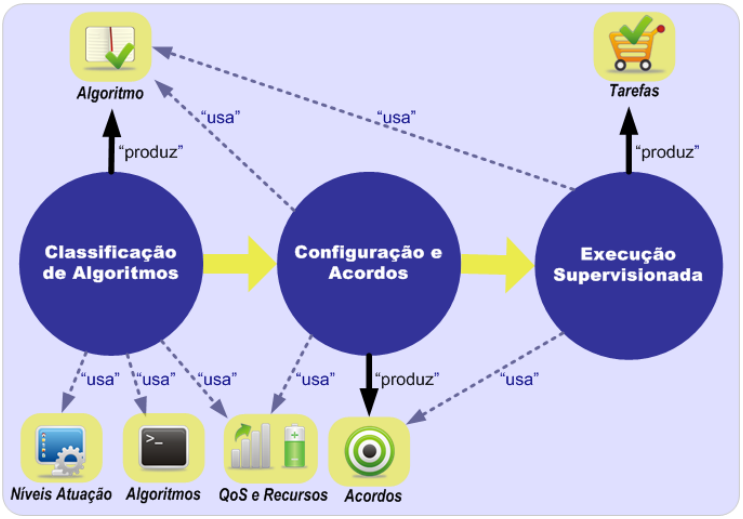

Figura 2. Metodologia - Implementação do *Módulo Conciliação*

A primeira ação executada pelo *Módulo Conciliação* é a filtragem dos algoritmos de resolução disponíveis, considerando somente aqueles que se encaixam aos tipos de conflitos enfrentados pela aplicação. Em seguida, o módulo executa as seguintes ações, conforme ilustrado pela figura 2:

Classificação de Algoritmos: considerando as demandas e limites da aplicação para QoS e consumo de recursos, essa ação escolhe o melhor algoritmo de conciliação disponível no repositório. A demanda por QoS e as limitações de recursos das aplicações também são consideradas como contextos, devendo ser coletadas e providas à metodologia, para que a mesma possa compará-las aos meta-dados dos algoritmos. Neste trabalho, a implementação da ação de classificação de algoritmos para a metodologia foi realizada por meio de um sistema baseado em regras, descrito pela tabela 1. As regras são genéricas o suficiente para serem executadas por qualquer aplicação coletiva, ubíqua e ciente de contexto.

Configuração e Acordos: o algoritmo de conciliação selecionado deve algumas vezes ser ajustado para executar dentro das expectativas de QoS e da disponibilidade de

Tabela 1. Regras Implementadas para a Metodologia Proposta

Pré-condições			**Algoritmo Selecionado**
Bateria	Perfis	Rede	
$\geq 50\%$	homogêneos	–	Maior QoS Coletivo
$\geq 50\%$	heterogêneos	–	Maior QoS Individual
$< 50\%$ & $> 25\%$	–	baixa qualidade	Menor Impacto na Rede
$< 50\%$ & $> 25\%$	–	–	Maior QoS Coletivo/Individual
$\leq 25\%$	–	–	Menor Consumo Energia

recursos da aplicação. Por exemplo, uma versão simplificada de algum algoritmo deve ser utilizada devido a limites para o consumo de energia e memória dos dispositivos dos usuários. O algoritmo selecionado é ajustado se possuir parâmetros variáveis que possam ser modificados. Um conjunto de acordos é em seguida formulado, considerando as características da versão final do algoritmo, de maneira a garantir que o mesmo funcione dentro das expectativas da aplicação.

Execução Supervisionada: uma vez selecionado e preparado, o algoritmo inicia a sua execução. Essa execução é supervisionada para garantir que os acordos previamente definidos não serão ultrapassados. No caso da violação de um acordo, o módulo de conciliação pode, por exemplo, abortar a execução do algoritmo e iniciar o processo novamente ou continuar a execução aplicando no entanto algum tipo de penalidade ao algoritmo. Se a segunda opção for implementada, a penalidade influenciará na escolha do algoritmo em futuras execuções.

Conforme mencionado anteriormente, a implementação dos módulos proposta acima é flexível e parametrizável. Isso significa que basta que cada aplicação forneça para a metodologia alguns valores de parâmetros, para que a mesma adapte seus algoritmos e funcione de acordo com os tipos de conflitos coletivos relacionados. Para que as aplicações possam prover à metodologia todas as informações necessárias ao correto funcionamento dos módulos de detecção e resolução de conflitos conforme propostos acima, i.e., para parametrizar esses módulos, elas devem utilizar um conceito definido neste trabalho, chamado de *Nível de Atuação*.

Definição 5 (Nível de Atuação) *Informações providas pela aplicação aos módulos da Conflict Engine que indicam como o módulo de detecção procura por conflitos e como o módulo de conciliação os resolve.*

Cada aplicação deve definir e utilizar seu próprio *Nível de Atuação*, o qual é passado em tempo de execução para a metodologia. Neste trabalho, o *Nível de Atuação* é provido aos módulos por meio de um arquivo de configuração XML (*eXtensible Markup Language*) contendo as seguintes informações: dimensões consideradas para verificação de conflitos, parâmetros específicos relacionados a cada uma delas, lista de algoritmos de resolução e valores iniciais para os meta-dados relacionados.

A próxima seção descreve a avaliação de dois estudos de caso, cada um deles considerando uma aplicação coletiva, ubíqua e ciente de contexto com características diferentes. A avaliação foi realizada dessa maneira com o objetivo de demonstrar como a mesma implementação para os módulos de detecção e resolução de conflitos pode ser utilizada para aplicações distintas, bastando que as mesmas forneçam seus *Níveis de Atuação*.

5. Avaliação

A avaliação da arquitetura proposta para resolução de conflitos coletivos foi realizada por meio de simulações utilizando o simulador Siafu [Martin and Nurmi 2006]. Dentre algumas opções encontradas na literatura, o Siafu foi escolhido pela sua robustez, flexibilidade e aderência aos conceitos da computação ciente de contexto. O Siafu é um simulador livre e de código aberto (possui licença GNU GPL) escrito na linguagem de programação Java e que possui suporte ao desenvolvimento de aplicações cientes de contexto multi-usuários, necessário para este trabalho. A arquitetura proposta foi implementada em Java, na forma de um anexo ao simulador Siafu.

Os resultados apresentados nos gráficos e tabelas representam valores médios, obtidos em 33 rodadas de simulação. Os valores para as fontes de energia utilizados foram escolhidos empiricamente de maneira a proporcionar uma degradação contínua da fonte de energia dos dispositivos até o final da execução das aplicações, sem contudo causar o esgotamento completo das mesmas. O consumo de energia com processamento foi determinado a partir da complexidade dos algoritmos implementados. O consumo de energia com comunicação não foi considerado nos estudos de caso visto que, com a utilização de uma arquitetura centralizada, ele seria o mesmo para qualquer um dos algoritmos do repositório.

5.1. Diretrizes

Os experimentos realizados avaliam o desempenho dos módulos do servidor da arquitetura proposta, considerando a implementação da metodologia descrita na seção 4.3, comparada aos resultados obtidos por outros cenários cujos módulos de conciliação utilizam sempre o mesmo algoritmo para tratar os conflitos detectados.

Os algoritmos utilizados em ambos os estudos de caso, bem como os meta-dados relacionados aos mesmos, estão descritos na tabela 2, onde x denota a capacidade máxima de carga de uma tarefa, n é o número total de usuários e m é um subconjunto de n ($m < n, m \geq x$). As métricas utilizadas para avaliar os cenários foram o consumo de energia dos dispositivos participantes e o nível de satisfação coletiva dos usuários. Todas as simulações começam com a execução de um passo de aquecimento e treinamento para o aprendizado do percentual de satisfação média de cada algoritmo. A seleção de algoritmos realizada pelos cenários que implementam a arquitetura proposta utiliza o sistema baseado em regras descrito na seção 4.3.

5.2. Estudo de caso 1: Atendimento de Emergência em Desastres de Larga Escala

Em cenários de atendimento de emergência em desastres de larga escala, tais como a passagem de furacões ou acidentes aéreos, é comum existirem diversas vítimas com diferentes perfis pessoais e níveis de lesão. Em tais situações, uma equipe de resgate é contactada e enviada para o local do desastre com equipamentos (ambulâncias, helicópteros,

Tabela 2. Repositório de Algoritmos para Resolução de Conflitos Coletivos

Algoritmo	Descrição	Complexidade	Satisfação
Sequencial (S)	Seleciona primeiros x usuários	$O(m)$	Baixa
Prioridades (P)	Seleciona x usuários de maior prioridade	$O(n) + O(m)$	Alta/Média
Prioridades Limitadas (BP)	Seleciona x usuários de todas as prioridades, começando pelas maiores	$O(n) + O(x)$	Média
Maioria (M)	Seleciona a tarefa mais indicada	$O(nm)$	Alta
Aleatório (A)	Seleciona a tarefa indicada por usuário aleatório	$O(1)$	Baixa

dentre outros), para executar os primeiros socorros. Cada uma das várias vítimas envolvidas, apesar de sua condição de saúde comparada à dos demais, deve ser atendida o mais rapidamente e precisamente possível. Entretanto, médicos e enfermeiras atribuídos para cuidar dos pacientes não conseguem atender a todos ao mesmo tempo. Além disso, ambulâncias e hospitais possuem uma capacidade máxima de carga que deve ser respeitada.

O objetivo da aplicação ciente de contexto desenvolvida é ajudar a decidir quais recursos (médicos, enfermeiras, ambulâncias, leitos em hospitais) serão alocados para quais pacientes em um dado momento específico. Basicamente, cada paciente deve passar por três passos de atendimento de emergência: assistência médica local, alocação em ambulância e admissão em um hospital. A aplicação provê algumas tarefas para cada um dos passos, e os pacientes devem executar no máximo uma em cada caso. No entanto, cada tarefa possui uma capacidade de carga máxima, relacionada à disponibilidade de seus recursos relacionados. Assim, conflitos ocorrem quando o número de usuários que precisam executar simultaneamente uma dada tarefa é maior do que a sua capacidade máxima. A configuração do *Nível de Atuação* dessa aplicação indica que conflitos ocorrem devido a incompatibilidades entre as demandas dos usuários e a disponibilidade de recursos do ambiente. Somente os algoritmos Sequencial, Prioridades e Prioridades Limitadas atendem à este nível de atuação. Quatro cenários foram projetados e simulados:

- *Cenário 1*: usa a metodologia proposta;
- *Cenário 2*: usa somente o algoritmo Sequencial;
- *Cenário 3*: usa somente o algoritmo Prioridades;
- *Cenário 4*: usa somente o algoritmo Prioridades Limitadas.

Foi considerado um grupo de 60 pacientes, cada um utilizando um dispositivo computacional móvel com 25 Joules de bateria, responsável por coletar informações contextuais. Quinze diferentes tarefas foram disponibilizadas aos usuários, sendo cinco para cada um dos passos de atendimento de emergência. A qualidade de serviços é medida pelo percentual de satisfação médio obtido pelos usuários considerando a seleção final de pacientes para executar as tarefas coletivas. Um usuário é considerado satisfeito se, em caso de conflito, o mesmo é selecionado pelo sistema para executar a tarefa indicada. Um valor de prioridade foi associado aleatoriamente a cada usuário para ser adotado pelos algoritmos que necessitam dessa informação.

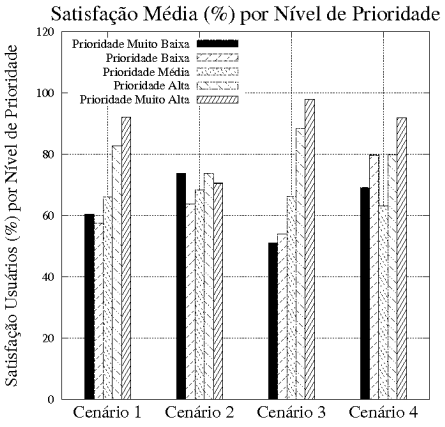

Figura 3. Estudo de Caso 1 - Satisfação por Prioridades

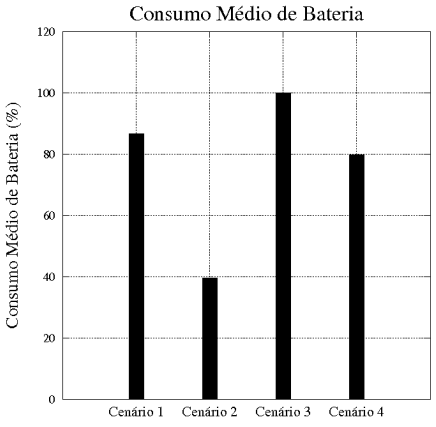

Figura 4. Estudo de Caso 1 - Consumo Médio de Energia

A figura 3 mostra o percentual de satisfação média por nível de prioridade. O *Cenário 2* não considera as prioridades dos usuários e portanto não possui qualquer padrão para este percentual. O *Cenário 3* está sempre preocupado em dar assistência primeiro aos usuários de maior prioridade, conforme claramente mostrado pela figura. Entretanto, uma vez que esse cenário não considera o consumo de recursos, os dispositivos dos usuários podem ficar sem bateria antes que todo o atendimento de emergência ao desastre esteja finalizado. Um comportamento intermediário é apresentado pelo *Cenário 1*, que mantém quase o mesmo padrão de satisfação que o *Cenário 3*, mas também tenta economizar os recursos dos dispositivos, sempre que possível, mantendo-os em operação até o final do atendimento. O *Cenário 4* tenta balancear a satisfação dos usuários por todas as prioridades possíveis. Entretanto, seu percentual de satisfação média para os usuários de maior prioridade é menor do que aqueles apresentados pelos *Cenários 1* e *3*.

A figura 4 apresenta o consumo médio de energia por cenário simulado. A avaliação dessa métrica mostra que, apesar de apresentar o melhor percentual médio de satisfação para os usuários de maior prioridade, o *Cenário 3* consome a maior quantidade de energia. Isso pode impedir a completa operação do atendimento de emergência durante todo o tempo de vida da aplicação. O consumo de energia no *Cenário 1* é apenas 5% maior do que o do *Cenário 4* e 15% menor do que o do *Cenário 3*. Mais uma vez, um comportamento intermediário do *Cenário 1* é evidente. O *Cenário 2* apresenta o menor consumo de energia. Todavia, ele possui o pior percentual médio de satisfação para os usuários de maior prioridade.

A figura 5 mostra a satisfação média para os usuários de prioridade muito alta, considerando diferentes números de usuários e os quatro cenários simulados. Em todos os cenários, a satisfação diminui enquanto o número de usuários aumenta. Os *Cenários 2* e *3* apresentam respectivamente a pior e a melhor média de satisfação considerando qualquer número de usuários. Os *Cenários 1* e *4* apresentam um comportamento intermediário. No entanto, enquanto o último apresenta uma taxa regular de diminuição da satisfação média, o primeiro é claramente mais similar ao *Cenário 3* enquanto o número de usuários é menor do que 60, mudando para o comportamento do *Cenário 2* caso contrário.

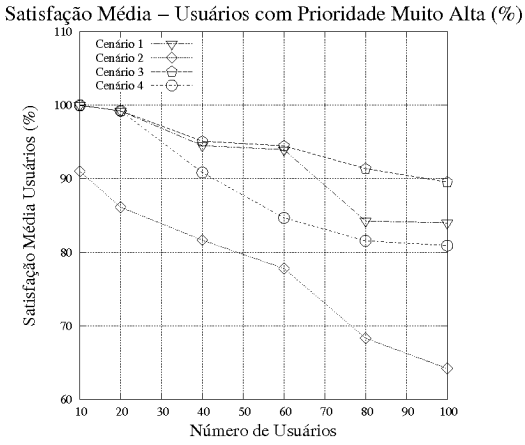

Figura 5. Estudo de Caso 1 - Satisfação Maior Prioridade

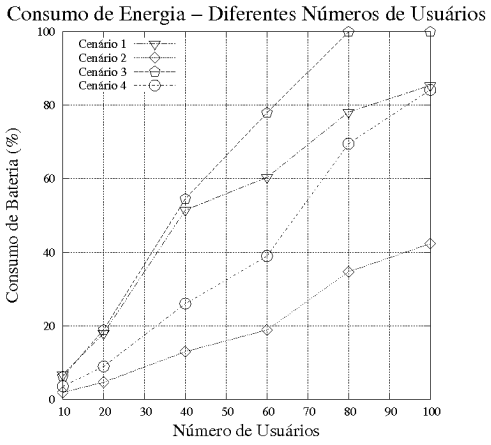

Figura 6. Estudo de Caso 1 - Percentual Consumo de Energia

Finalmente, a figura 6 mostra o percentual médio de consumo de energia para diferentes números de usuários. Fica claro que o consumo aumenta com o número de usuários, uma vez que o número de conflitos é maior e, portanto, a aplicação terá que executar algoritmos de conciliação mais frequentemente. O *Cenário 3* alcança 100% de consumo com 80 usuários ou mais. É também possível observar como o *Cenário 1* opera similarmente ao *Cenário 3* quando o número de usuários é menor do que 60, mudando para o padrão de operação do *Cenário 2* com um número maior de usuários. Apesar de gastar menos energia, o *Cenário 4* apresenta menor satisfação média para usuários de maior prioridade, conforme mostrado pela figura 5.

5.3. Estudo de caso 2: Guia Turístico Coletivo

Um guia turístico coletivo também foi implementado como um estudo de caso para a arquitetura proposta neste trabalho. Essa aplicação considera um grupo de turistas em um passeio de um dia, executando tarefas com diferentes características. Nesse cenário, a execução de tarefas pode ser bastante afetada pelos contextos ambientais coletados, bem como pelos perfis de cada usuário. Um grupo de 50 usuários compartilhando uma ferramenta de guia turístico computacional foi considerada e, antes de executar qualquer tarefa, cada participante indica qual seria sua tarefa preferida naquele momento. O servidor recebe todas as tarefas indicadas e executa os módulos para verificação e resolução de conflitos da *Conflict Engine*. O *Nível de Atuação* dessa aplicação indica que conflitos ocorrem quando usuários selecionam diferentes tarefas em uma dada rodada. Portanto, somente os algoritmos Prioridades, Maioria e Aleatório são considerados.

A qualidade de serviços é medida pelo percentual de satisfação médio obtido pelos usuários com a seleção final de tarefas a serem executadas. Um usuário é considerado satisfeito se a tarefa selecionada para ser executada for a mesma que ele inicialmente indicou. Foi considerado que cada usuário possui seu próprio dispositivo computacional móvel com uma bateria inicial de 10 joules. Oito tarefas diferentes foram disponibilizadas para os usuários. Três cenários foram projetados e simulados:

- *Conflict Engine*: usa a metodologia proposta;

- *Recursos*: usa somente o algoritmo Aleatório;
- *Satisfação*: usa somente o algoritmo Maioria.

As figuras 7 e 8 apresentam, respectivamente, *snapshots* da energia residual e percentual médio de satisfação dos usuários para cada rodada, durante uma execução da simulação. O cenário *Conflict Engine* possui um padrão de consumo de energia bastante similar ao cenário *Satisfação*. De fato, após a quinta rodada, ele apresenta valores de consumo menores. Todavia, considerando o *snapshot* para satisfação dos usuários, o cenário *Conflict Engine* apresenta porcentagem maior do que o do cenário *Recursos* em quase todas as rodadas.

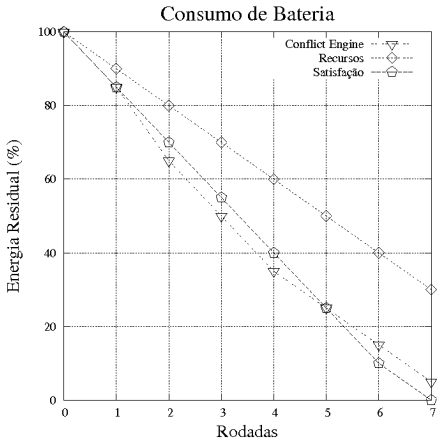

Figura 7. Estudo de Caso 2 - Energia Residual (*snapshot*)

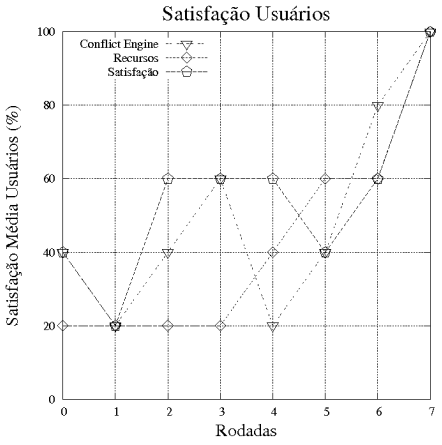

Figura 8. Estudo de Caso 2 - Satisfação Usuários (*snapshot*)

De acordo com a tabela 3, apesar do cenário *Recursos* apresentar a menor média de consumo de energia, e o cenário *Satisfação* a maior média para a satisfação dos usuários, o cenário *Conflict Engine* apresenta uma média de satisfação muito próxima daquela do cenário *Satisfação*, com uma menor média de consumo de energia. Em outras palavras, os usuários se sentiram satisfeitos com as tarefas selecionadas em casos de conflito e ainda apresentaram um consumo médio de energia interessante. Apesar de prover o melhor percentual de satisfação aos usuários, os dispositivos dos usuários do cenário *Satisfação* somente funcionam até a quinta rodada devido à falta de energia.

Tabela 3. Estudo de Caso 2 - Comparação Cenários

Cenário	*Conflict Engine*	*Recursos*	*Satisfação*
Consumo de Energia (%)	95.0	70.0	100.0
Satisfação Usuários (%)	50.0	42.0	55.0
Última Rodada	7	7	5

6. Conclusões e Trabalhos Futuros

Esse trabalho apresentou uma arquitetura centralizada e com rodízio de servidores para detecção e resolução de conflitos coletivos. As atividades executadas pelo servidor foram

definidas como módulos de um arcabouço genérico para aplicações cientes de contexto, e a implementação dos mesmos considerando sistemas ubíquos foi modelada como uma metodologia específica, que considera as restrições de recursos e as demandas por QoS desses sistemas. Duas aplicações diferentes foram avaliadas, mostrando que a metodologia foi capaz de manter a satisfação dos usuários e aumentar o tempo de vida da aplicação, economizando recursos sempre que necessário. Também foi demonstrado que a solução é flexível o suficiente para ser utilizada por diferentes aplicações. Como trabalhos futuros, pode-se listar a avaliação de outras aplicações e a implementação de novos algoritmos para a metodologia.

Referências

Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277.

Capra, L., Emmerich, W., and Mascolo, C. (2003). Carisma: context-aware reflective middleware system for mobile applications. *Software Engineering, IEEE Transactions on*, 29(10):929–945.

Dey, A. K. (2001). Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7.

Huebscher, M. C., McCann, J. A., and Hoskins, A. (4 December 2006). Context as autonomic intelligence in a ubiquitous computing environment. *International Journal of Internet Protocol Technology*, 2:30–39(10).

Martin, M. and Nurmi, P. (2006). A generic large scale simulator for ubiquitous computing. *Mobile and Ubiquitous Systems, Annual International Conference on*, 0:1–3.

Park, I., Lee, K., Lee, D., Hyun, S. J., and Yoon, H. Y. (2005). A dynamic context-conflict resolution scheme for group-aware ubiquitous computing environments. In *Proceedings of the 1st International Workshop on Personalized Context Modeling and Management for UbiComp Applications (ubiPCMM'05)*, pages 42–47, Tokyo, Japan.

Roy, N., Roy, A., and Das, S. K. (2006). Context-aware resource management in multi-inhabitant smart homes: A nash h-learning based approach. In *PERCOM '06: Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 148–158.

Shin, C., Dey, A. K., and Woo, W. (2008). Mixed-initiative conflict resolution for context-aware applications. In *UbiComp '08: Proceedings of the 10th international conference on Ubiquitous computing*, pages 262–271, New York, NY, USA. ACM.

Shin, C. and Woo, W. (2005). Conflict resolution method utilizing context history for context-aware applications. In *Proceedings of the 1st International Workshop on Exploiting Context Histories in Smart Environments (ECHISE'05)*, Munich, Germany.

Weiser, M. (1993). Some computer science issues in ubiquitous computing. *Commun. ACM*, 36(7):75–84.