

# Uma API Pub/Sub para Aplicações Móveis Sensíveis ao Contexto

Gustavo Baptista, Markus Endler, Hana Rubinsztein  
Pontifícia Universidade Católica  
do Rio de Janeiro (PUC-RJ)  
R. Marquês de São Vicente, 225  
22453-900, Rio de Janeiro, RJ  
{gbaptista,endler,hana}@inf.puc-rio.br

Vagner Sacramento  
Instituto de Informática  
Universidade Federal de Goiás (UFG)  
UFG - Bloco IMF I  
Campus II - Samambaia - Goiânia - GO  
vagner@inf.ufg.br

## Resumo

*Comunicação baseada em eventos é importante para aplicações sensíveis ao contexto pois permite a detecção de mudanças no meio de execução sem que seja necessária uma consulta periódica para obtenção do contexto corrente. É também o mecanismo de interação mais apropriado para aplicações móveis devido ao desacoplamento entre cliente/servidor oferecido pela comunicação assíncrona no modelo de eventos publish/subscribe. Para suportar a mobilidade de clientes, um serviço de notificação de eventos deve ser capaz de lidar com a desconexão temporária e a mobilidade de clientes, i.e. a entrega confiável de notificações independente do endereço IP corrente dos mesmos. Com o intuito de tratar destas questões, propõe-se uma API que oferece suporte à comunicação pub/sub para aplicações móveis sensíveis ao contexto. Esta API faz parte do middleware Mobile Collaboration Architecture (MoCA), desenvolvido na PUC-Rio, e tem sido utilizada por diversos serviços e aplicações deste middleware.*

## 1. Introdução

Além das características primárias que fazem parte da lógica de negócio, algumas aplicações levam em conta o contexto corrente do usuário (e.g., localização), dispositivo ou ambiente computacional (e.g., rede, sistemas operacionais, etc) para se adaptarem em função da mudança do contexto, e, por conseguinte proverem serviços mais adequados ao usuário final. Por exemplo, uma aplicação sensível ao contexto poderia usar as informações de conectividade da rede sem fio para adaptar o seu comportamento de acordo com a comunicação intermitente da rede, ou usar a informação de localização para enviar uma oferta de um produto de uma loja co-localizada com o usuário.

Serviços ou APIs para comunicação baseada em eventos são importantes para aplicações sensíveis ao contexto

porque estas aplicações precisam ser reativas quanto às mudanças no meio de execução, para que possam ajustar o seu modo de funcionamento apropriadamente. Além disso, o programador da aplicação não deve ter que se preocupar em fazer o “polling” (consulta periódica) para obter o contexto corrente de um dispositivo (e.g. memória disponível, nível da bateria, qualidade da conectividade, localização, etc.).

A comunicação baseada em eventos também é o mecanismo de interação mais apropriado para aplicações móveis em geral, devido ao desacoplamento entre cliente/servidor oferecido pela comunicação assíncrona no modelo de eventos publish/subscribe. Algumas limitações e restrições intrínsecas das redes sem-fio e dos dispositivos móveis tais como a mobilidade do usuário, a comunicação intermitente e a desconexão dos dispositivos podem ser tratadas por um serviço de eventos adequado para este cenário.

Os aspectos de mobilidade discutidos nos sistemas pub/sub apresentados na literatura [1, 2, 3, 5, 7, 10, 11, 14], pressupõe a existência de uma infra-estrutura de suporte ao endereçamento transparente à mobilidade (e.g., Mobile IP [9]), e têm portanto seu foco de discussão apenas em aspectos de mobilidade que afetam suas abordagens em um nível mais alto, tais como o roteamento de notificações em uma federação de servidores de eventos (i.e. rede de brokers) e outros problemas que o modelo pub/sub deve tratar quanto a mobilidade (e.g., persistência de notificações, políticas de entrega e etc.). No entanto, com essa abordagem, a mobilidade dos clientes torna-se transparente para o middleware, o que impede a execução de adaptações específicas e relativas à “localização” (i.e. ponto de conexão) do cliente.

Neste artigo, propõe-se uma API de comunicação baseada em eventos, chamada Event-based Communication Interface (ECI), que trata das questões de mobilidade para as aplicações pub/sub sem requerer nenhuma infra-estrutura adicional. Esta API faz parte do middleware Mobile Collaboration Architecture (MoCA) [?], desenvolvido na PUC-Rio, e tem sido utilizada por diversos serviços e aplicações

deste middleware.

### 1.1. MoCA

A *Mobile Collaboration Architecture* [13] é um middleware que têm o objetivo de dar suporte a aplicações móveis sensíveis ao contexto e consiste de serviços e APIs que dão suporte à aquisição, armazenamento e processamento de contexto, comunicação síncrona e assíncrona e um framework para a implementação de proxies de aplicações. A MoCA foi projetada para ser utilizada em uma rede LAN sem-fio infra-estruturada, e sua versão corrente roda em WinXP/CE e é baseada em TCP/IP. Alguns dos principais serviços da MoCA, responsáveis por obter, armazenar e disponibilizar informações de contexto computacional são:

**Monitor** é um daemon executando em cada dispositivo móvel que é encarregado de coletar dados relativos ao estado de execução/conectividade e enviar estes dados para o CIS (Context Information Service) executando em um (ou mais) nós da rede cabeada. Os dados coletados incluem a qualidade da conexão sem-fio, energia disponível, utilização de CPU, memória disponível, Access Point corrente dentre outras.

**Context Information Service (CIS)** é um serviço distribuído onde cada servidor CIS recebe e processa os dados de contexto dos dispositivos, enviados pelos seus respectivos Monitores. Ele utiliza o ECI para receber requisições de notificações (subscrições) com expressões de interesse sobre dados de contexto dos dispositivos, e entregar notificações para as aplicações quando a expressão de interesse correspondente atende um novo estado das variáveis de contexto.

## 2. Comunicação Publish/Subscribe

A principal característica da interação publish/subscribe é o desacoplamento espacial e temporal dos participantes da comunicação. Como eles interagem através de um intermediário, não necessitam se conhecer, e nem estar ativos simultaneamente para a troca de mensagens, ficando a cargo da infra-estrutura pub/sub o armazenamento e distribuição das mensagens. Esta comunicação assíncrona, desacoplada e persistente é importante para os dispositivos móveis que estão freqüentemente indisponíveis devido à conexão intermitente ou à potência limitada da bateria, e à mobilidade [?, 8]. Além disto, o modelo pub/sub suporta extensibilidade do sistema. A adição de um novo publicador ou de um assinante não afeta a funcionalidade do sistema, característica que é desejável ao lidar com as mudanças freqüentes do ambiente móvel.

Os sistemas publish/subscribe podem ser implementados de várias maneiras, cada uma delas mais apropriada a um tipo de aplicação. Duas características principais a se observar são: a forma como os assinantes especificam os eventos em que estão interessados (esquema de registro de interesse) e a arquitetura do serviço de eventos ou infra-estrutura de envio (ou seja, a topologia de interconexão).

### 2.1. Esquemas de Registros de Interesse

Estes sistemas também permitem que clientes especifiquem as características dos eventos que desejam receber, através de registros de interesse que são usados para filtrar os eventos de acordo com a necessidade dos assinantes. Os registros contém regras que permitem ao serviço pub/sub comparar cada evento publicado com o interesse de cada assinante, e entregar apenas as notificações de eventos que obedeçam ao critério definido no registro. Os principais esquemas de definição de registros de interesse (e filtragem de eventos) utilizados pelos sistemas pub/sub são baseados em tópicos, em conteúdo ou em tipos.

O esquema baseado em tópicos foi o primeiro esquema utilizado por sistemas pub/sub, e é baseado na noção de tópicos ou assuntos. Os participantes podem publicar eventos e registrar interesse em tópicos, identificados por palavras-chave. Na prática, o esquema provê abstrações que mapeiam tópicos em canais de comunicação distintos.

O esquema baseado em conteúdo utiliza uma abordagem em que os registros de interesse são descritos em termos do conteúdo ou das propriedades dos eventos publicados. Estas propriedades podem ser atributos internos das estruturas de dados dos eventos, como em Siena [2], Elvin [14, 15] e Jedi [4], ou meta-dados associados aos eventos, como em JMS [6]. O registro de interesse especifica filtros, através de uma linguagem própria. Os filtros definem restrições, geralmente na forma de pares nome-valor das propriedades e operadores de comparação (=, <, <=, >, >=), que identificam os eventos válidos. Estas restrições podem ser logicamente combinadas ( com operadores lógicos or, and e etc) para formar padrões complexos de registro de interesse.

### 2.2. Modelos de Arquitetura

A implementação de um serviço de notificação de eventos utilizando publish/subscribe pode seguir uma abordagem centralizada [14] onde um único elemento

é responsável por manter o registro de interesse dos assinantes e por disseminar os eventos que são gerados. Em outra abordagem, adotada em [12, 17], a comunicação assíncrona é realizada através de primitivas que implementam mecanismos de envio não bloqueantes, tanto em publicadores como em assinantes, sem a necessidade de um intermediário. Nesta arquitetura descentralizada, todos os nós são homogêneos, têm um poder de processamento razoável, e estão aptos a gerenciar os registros de interesse e disponibilizar funções pub/sub. Outra abordagem é uma arquitetura distribuída que compreenda uma rede de servidores especiais, denominados *brokers*. Cada broker é responsável por prover o serviço pub/sub para parte dos clientes, e por executar os protocolos necessários para persistência, confiabilidade, ou alta-disponibilidade, bem como filtragem e roteamento de eventos baseado em conteúdo. Essa rede de brokers distribuídos pode ser interligada sob diferentes topologias.

### 2.3. Suporte a Mobilidade e Desconexão

Suporte à mobilidade é a capacidade dos clientes móveis migrarem entre diferentes redes (domínios, ou redes com tecnologias de comunicação diferentes) durante a utilização de algum serviço. A maioria dos sistemas publish/subscribe existentes foi projetada para ambientes estacionários, e as operações relacionadas a mobilidade são tratadas pela aplicação. Por exemplo, a aplicação deve se encarregar de efetuar o cancelamento de registro no sistema pub/sub antes da desconexão e novamente se registrar após a reconexão, mas neste caso o assinante não receberá eventos publicados durante o tempo de desconexão.

Em [11, 16], argumenta-se que o próprio middleware pub/sub deve oferecer suporte à mobilidade e assegurar reconexões transparentes aos clientes e que preservem as notificações publicadas durante a desconexão. Os problemas relacionados à mobilidade têm sido tratados recentemente [1, 5, 16]. Boa parte das soluções propostas para a mobilidade do cliente em middlewares pub/sub não são otimizadas para ambientes móveis, mas apenas estendem os sistemas estacionários existentes [1, 5].

A solução mais comum é baseada na abordagem de “enfileiramento” de eventos, na qual um nó especial (broker) da infra-estrutura do sistema que é responsável por servir diversos clientes (geralmente o último que serviu um assinante) atua como um proxy do assinante durante sua desconexão. Este proxy armazena as notificações publicadas durante a desconexão do assinante em uma fila especial, e as entrega

após a reconexão deste assinante. Se o assinante se reconectar ao sistema através de um novo broker, é executado um procedimento de handoff que transfere as notificações armazenadas ao assinante e atualiza as rotas da rede de brokers.

Outra abordagem é o uso de notificações persistentes [11, 6], que ao invés de armazenar as notificações em filas especiais para cada cliente, a rede de brokers armazena as notificações persistentes até seu prazo de validade expirar. Os publicadores devem definir o período de validade das notificações, e o sistema deve garantir o armazenamento das notificações durante seu período de validade, e entregar notificações válidas aos assinantes quando estes se reconectam ao sistema publish/subscribe. Se o assinante se reconectar após a expiração da notificação, esta não é entregue pois se considera que ela não possui mais informações válidas.

### 3. Event Based Communication Interface

O ECI segue o modelo de um serviço de eventos baseado em tópicos combinado com um esquema de filtragem baseado em propriedades dos eventos, que permite que qualquer aplicação ou serviço torne-se um publicador de eventos ou que aceite subscrições de notificações relativas à mudança de propriedades de contexto associadas a um tópico. Ela é um componente importante do Middleware MoCA, pois faz parte do Serviço de Informações de Contexto (CIS, Context Information Service) e é utilizada como base para a disseminação de informações de contexto dos dispositivos móveis, porém é também uma API pub/sub genérica que pode ser utilizada por qualquer outra aplicação.

O ECI utiliza uma combinação do esquema de registro baseado em tópicos com o esquema baseado em conteúdo. Desta forma a API provê a aceitação de subscrições de notificações relativas à mudança de propriedades de contexto associadas a um tópico, o que é muito interessante para sua utilização como disseminador de dados de contexto associados a um tópico. Por exemplo, a MoCA faz uso desta característica definindo um tópico como o identificador de um dispositivo (i.e. MAC address) e as propriedades (conteúdo) dos eventos como seus dados de contexto no momento da publicação (i.e. porcentagem de utilização de CPU, memória disponível, *Access Points* no alcance do dispositivo, coordenadas GPS). Uma aplicação que deseja monitorar um dispositivo, se registra em um tópico, o identificador do dispositivo, especificando o filtro sobre os dados de contexto (propriedades dos eventos) que deseja receber, como por exemplo a expressão

((EnergyLevel < 65) OR (FreeMemory < 18000)) definiria interesse por eventos em que a energia do dispositivo é menor que 65% ou a quantidade de memória disponível é inferior a 18000 KBs.

O ECIClient é a classe que implementa o cliente de eventos e que pode ser instanciada por uma aplicação cliente para se registrar em um ECIServer como interessada em receber notificações da ocorrência de eventos/mensagens relacionados a um assunto. Ela também pode ser utilizada para a publicação de eventos para os demais clientes registrados no ECIServer.

Para se registrar como interessada em um evento, uma aplicação cliente pode utilizar diversos tipos de métodos para subscrição. Por exemplo, o método `subscribe(String subject, String expression)` envia uma Subscrição (*subscription*) para o ECIServer notificando o interesse por eventos relacionados a um assunto (*subject*) e uma expressão condicional que é utilizada para filtrar os eventos de interesse de acordo com suas propriedades. Esta expressão booleana é avaliada de acordo com as propriedades do evento enviado pelo seu publicador. Este método retorna um tópico (*Topic*) que representa o assunto de interesse. O tópico deve ser utilizado para adicionar ou remover listeners de eventos no ECIClient ou para remover a subscrição sobre o tópico do ECIServer. O cliente também pode especificar o número máximo de vezes em que deve ser notificado enquanto a condição (expressão) do evento for atendida repetidas vezes, ou seja, quantas vezes a expressão foi validada sem que uma negação tenha ocorrido. O cliente será notificado nas próximas *n* vezes (de acordo com o limite especificado) que um evento relacionado ao assunto e a expressão ocorreu. Quando o servidor detecta que uma negação da expressão de interesse ocorreu (as propriedades do evento não correspondem à expressão de interesse) o contador de notificações é reinicializado para zero e a partir de então o cliente voltará a ser notificado *n* vezes nas novas ocorrências do evento.

Uma característica interessante das subscrições com expressões no ECI, é a capacidade do cliente especificar que deseja ser notificado também (apenas uma vez) quando a expressão de interesse não é atendida (i.e. se torna falsa). Isto pode ser utilizado para algumas aplicações especiais que necessitam detectar quando uma condição não é mais válida. Por exemplo, uma aplicação pode enviar uma subscrição pedindo para ser notificada quando o estado de um dispositivo corresponde a expressão (EnergyLevel < 25); Quando esta propriedade é verdadeira a aplicação deverá receber alguns eventos de notificação e pode implementar alguma adaptação (se necessário) para prover um serviço

mais apropriado para o estado atual do dispositivo, no caso, a carga da bateria estaria baixa e alguma providência poderia ser tomada pela aplicação para diminuir o consumo de energia. Entretanto, a aplicação precisa também ser notificada quando o estado do dispositivo mudou, ou seja, neste caso a bateria foi recarregada, para que ela possa parar de implementar tal adaptação.

Para remover uma subscrição feita no ECIServer, a aplicação pode chamar o método `unsubscribe(Topic topic)`, o tópico representando o assunto/expressão de interesse e que foi obtido através de uma chamada ao método `subscribe`, que subscreveu a aplicação no servidor.

O ECIServer é a classe que implementa o servidor de eventos e que pode ser instanciada por uma aplicação para prover uma interface de comunicação `publish/subscribe` para diversos clientes, controlando as subscrições dos clientes e notificando as ocorrências de eventos/mensagens. Ele é responsável por receber e processar todas as requisições feitas pelos clientes (ECIClients) tais como subscrições, eventos, requisições e notificações de controle. Os clientes podem registrar aplicações no ECIServer como interessadas em receber notificações de eventos/mensagens relacionados a um assunto através da classe ECIClient, que é instanciada em cada aplicação que participará como cliente do serviço.

### 3.1. Publicação de eventos

Tanto o ECIServer como o ECIClient podem ser utilizados para a publicação de eventos/mensagens para um determinado assunto. O ECIServer é utilizado para a publicação por uma aplicação servidora quando esta aplicação necessita publicar um evento/mensagem diretamente para seus clientes. O ECIClient é utilizado para a publicação por uma aplicação cliente quando esta aplicação precisa publicar um evento/mensagem para os outros clientes. Neste caso, o ECIServer age como um intermediador da comunicação entre os clientes. Para publicar um evento para os demais clientes alguns tipos de métodos do ECIClient podem ser utilizados, como por exemplo o método `publish(String subject, EventProperties eventProperties, Object data)` publica um evento para um assunto específico com propriedades para o evento publicado, que serão utilizadas como base para avaliar as expressões de interesse definidas por aqueles que se registraram. As propriedades provêm um mecanismo eficiente para suportar uma filtragem

de eventos definida pela aplicação. Elas permitem que uma aplicação filtre eventos de seu interesse utilizando critérios específicos de conteúdo para a aplicação. As propriedades de eventos podem ser dos tipo boolean, int, long, double e String. Por exemplo, se as propriedades de um evento forem definidas como `EnergyLevel=20; OnLine=true; FreeMemory=250000; Name='John'`; apenas os clientes que se registraram como interessados em eventos relacionados ao mesmo assunto e que especificaram uma expressão que corresponde aos valores das propriedades do evento publicado receberão uma notificação, no caso, uma expressão que seria satisfeita pelas propriedades acima seria `“(EnergyLevel <= 20 and OnLine) or (FreeMemory < 200000) or (Name = 'John')”`. Como já foi mencionado, tanto o ECIServer como o ECIClient podem ser utilizados para a publicação de eventos/mensagens para um determinado assunto. Na aplicação que utiliza o ECIServer, após a criação de uma instância desta classe, a aplicação servidora de eventos será capaz de publicar dados diretamente para os assinantes utilizando o métodos de publicação tais como aqueles existentes no ECIClient, definindo o assunto e as propriedades para o evento publicado.

### 3.1.1 Intercepção de eventos e publicação para clientes específicos

Algumas aplicações/serviços precisam interceptar os eventos publicados antes que suas notificações sejam entregues para os clientes interessados. Alguns exemplos deste tipo de caso seriam um serviço que realiza alguma adaptação sobre o conteúdo das mensagens ou controle de permissões ou privacidade. Para interceptar os eventos publicados em um ECIServer, a aplicação deve chamar o método `addPublicationListener (PublicationListener listener, boolean interceptPublications)` publicações (`PublicationListener`) e especificar que deseja interceptar os eventos publicados. A aplicação pode também adicionar um listener permitindo que os eventos sejam entregues, sem interceptação.

Além disso, para algumas aplicações interceptadoras de eventos publicados, pode também ser necessário publicar eventos especificamente para determinados clientes. Por exemplo, um serviço que intercepta eventos publicados, pode escolher quais clientes podem receber um determinado evento e publica-lo apenas para estes clientes.

## 3.2. Gerenciamento de Clientes e Validação de Subscrições

O ECIServer mantém registrados todos os clientes que possuem subscrições. Cada cliente possui um identificador único, i.e. uma string que consiste da concatenação do ip e porta de onde a subscrição do cliente foi recebida.

Em uma rede sem fio, clientes móveis podem possivelmente se desconectar (ou serem desligados) por longos períodos, sem antes remover as suas subscrições, o que pode gerar um acúmulo de subscrições não desejado no servidor, de subscrições que não sejam de nenhuma aplicação ativa. Para evitar este acúmulo, é realizada uma coleta de lixo de subscrições que não são validadas após um determinado período de tempo. Para isto, os clientes que possuem subscrições no servidor enviam notificações (`Advertisements`) periódicas indicando que estão ativos, permitindo que o servidor revalide todas as subscrições de clientes dos quais recebeu um `Advertisement`.

## 3.3. Suporte à Mobilidade e Desconexão de Clientes

A fim de permitir que o ECIServer suporte a mobilidade dos clientes que estão subscritos, entregando corretamente as notificações quando os clientes mudam de endereço, podemos aproveitar a validação periódica das subscrições realizadas pelos clientes (envio periódico de `Advertisements`) e incluir dentro da mensagem do `Advertisement` o endereço ip e porta correntes do cliente em questão. Como os clientes são identificados no servidor pela concatenação de seus ips e portas, no caso da mudança de endereço (mudança de ip e/ou porta) além do novo endereço ter que ser enviado para o servidor, faz-se necessário também, durante o processo de mudança de endereço, enviar o endereço antigo para que o servidor possa ser atualizado corretamente. Para evitar isso, podemos ao invés de utilizar a concatenação do ip e porta de cada cliente como identificador único, utilizar outra forma de identificação única para cada cliente, como por exemplo a concatenação do endereço MAC do dispositivo com um número aleatório persistente no dispositivo do cliente. O ip e porta correntes de um cliente são então considerados como atributos temporários de cada cliente registrado no servidor e desta forma mais facilmente atualizados no caso de uma mudança no endereço de entrega das notificações.

Quando os clientes realizam uma subscrição em um tópico, eles devem escolher políticas de entrega para

o caso da ocorrência de desconexões, tais como o armazenamento temporário das notificações importantes para entrega posterior ou descartar notificações não importantes ou que não farão sentido de ser recebidas após um certo período de tempo.

Quando ocorre uma notificação para um cliente do qual o ECIServer não recebeu Advertisements por um certo período de tempo, o ECIServer define o status do cliente em questão como desconectado e passa a executar a política de desconexão definida pelo cliente na subscrição, tal como o armazenamento temporário ou descarte das notificações.

### 3.4. Implementação

O ECI foi implementado para funcionar tanto em dispositivos móveis, tais como computadores de mão, smartphones e etc, como em computadores com maiores recursos como laptops ou desktops. Por ser implementado em Java, ele pode ser utilizado em qualquer dispositivo/sistema operacional que disponha de uma máquina virtual que seja compatível com a versão 1.4 desta linguagem. Para sua utilização em dispositivos portáteis (i.e computadores de mão) é necessária uma máquina virtual Java que implemente a especificação CDC 1.1 (Connected Device Configuration) da Sun Microsystems para máquinas virtuais.

### 4. Conclusões

Apresentamos o ECI, uma API que dá suporte à implementação de aplicações móveis sensíveis à contexto. O ECI está em sua versão 1.1.2 já é utilizado por muitos serviços de contexto da MoCA, tais como o Context Information Service, o Proxy Framework, o Context Privacy Service e outras aplicações móveis sensíveis a contexto e/ou que necessitam de comunicação pub/sub. Sua utilização demonstrou bom desempenho inclusive para centenas de clientes. O suporte a desconexão e à mobilidade está atualmente sendo implementado.

### Referências

- [1] M. Caporuscio, A. Carzaniga, and A. L. Wolf. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. *IEEE Transactions on Software Engineering*, 29(12):1059–1071, Dec. 2003.
- [2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3):332–383, 2001. Siena.

- [3] G. Cugola, E. D. Nitre, and G. Picco. Content-based dispatching in a mobile environment. 2001.
- [4] G. Cugola, E. D. Nitro, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. *IEEE Trans. Softw. Eng.*, 27(9):827–850, 2001.
- [5] L. Fiege, F. C. Gartner, O. Kasten, and A. Zeidler. Supporting mobility in content-based publish/subscribe middleware. In *Proceedings of the ACM/IFIP/USENIX International Middleware Conference (Middleware 2003)*, volume 2672 of Lecture Notes in Computer Science, pages 103–122. Springer-Verlag, June 2003.
- [6] M. Hapner, R. BurrIDGE, R. Sharma, J. Fialli, and K. Stout. *Java Message Service*. Sun Microsystems, April 2002.
- [7] Y. Huang and H. Garcia-Molina. Publish/subscribe in a mobile environment. *Wireless Networks*, pages 643–652, 2004.
- [8] H.-A. Jacobsen. Middleware services for selective and location-based information dissemination in mobile wireless networks. In *Workshop on Middleware for Mobile Computing. Middleware 2001*, pages 1–5, Heidelberg, Germany, Nov 2001.
- [9] D. Johnson. Scalable support for transparent mobile host internetworking. *Wireless Networks*, pages 311–321, 1995.
- [10] G. Mühl, A. Ulbrich, K. Herrmann, and T. Weis. Disseminating information to mobile clients using publish-subscribe. *IEEE Internet Computing*, 8(3):46–53, May/June 2004.
- [11] I. Podnar, M. Hauswirth, and M. Jazayeri. Mobile push: Delivering content to mobile users. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*, pages 563–570, Washington, DC, USA, 2002. IEEE Computer Society.
- [12] A. I. T. Rowstron, A. Kermarrec, M. Castro, and P. Druschel. Scribe: The design of a large-scale event notification infrastructure. *NETWORKED GROUP COMMUNICATION*, pages 30–43, 2001.
- [13] V. Sacramento, M. Endler, H. K. Rubinsztein, L. S. Lima, K. Goncalves, F. N. Nascimento, and G. A. Bueno. Moca: A middleware for developing collaborative applications for mobile users. *IEEE Distributed Systems Online*, 5(10):2, October 2004.
- [14] B. Segall, D. Arnold, J. Boot, M. Henderson, and T. Phelps. Content based routing with elvin4. In *Proceedings AUUG2K*, Canberra, Australia, June 2000.
- [15] P. Sutton, R. Arkins, and B. Segall. Supporting disconnectedness - transparent information delivery for mobile and invisible computing. In *CCGrid 2001 IEEE International Symposium on Cluster Computing and the Grid*, May 2001. Elvin.
- [16] A. Zeidler and L. Fiege. Mobility support with REBECA. In *Proceedings of the 23rd International Conference on Distributed Computing Systems - Workshops (ICDCS 2003 Workshops)*, pages 354–360, May 2003.
- [17] S. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. KATZ, and J. . Kubiawicz. Bayeux: an architecture for scalable and faulttolerant wide-area data dissemination. In *NOSSDAV*, pages 11–20, 2001.