PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 03/13

Desenvolvendo Aplicações de Rastreamento e Comunicação Móvel usando o Middleware SDDL

Igor Oliveira Vasconcelos
Rafael Oliveira Vasconcelos
Gustavo Luiz Bastos Baptista
Caio Pimentel Seguin
Markus Endler

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22451-900

RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 03/13 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Março, 2013

Desenvolvendo Aplicações de Rastreamento e Comunicação Móvel usando o Middleware SDDL

Igor Oliveira Vasconcelos Rafael Oliveira Vasconcelos
Gustavo Luiz Bastos Baptista Caio Pimentel Seguin

Markus Endler

{ivasconcelos, rvasconcelos, gbaptista, cseguin, endler}@inf.puc-rio.br

Abstract. This article explains, and gives some details, of how an application prototye for mobile tracking and communication among roadside inspectors and vehicles called Acompanhamento Remoto de Fiscais e Frotas (ARFF) was developed using the Scalable Data Distribution Layer(SDDL) middleware.

Keywords: distributed systems, mobile computing, middleware.

Resumo. Este artigo explica, e dá alguns detalhes, de como um protótipo de aplicação para monitoramento móvel e comunicação entre fiscais e veículos em uma estrada chamado Acompanhamento Remoto e Fiscais Frotas (ARFF) foi desenvolvido utilizando middleware Scalable Data Distribution Layer (SDDL).

Palavras-chave: sistemas distribuídos, computação móvel, middleware.

Responsável por publicações:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22451-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3527-1516 Fax: +55 21 3527-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

Sumário

1 Introdução

Os avanços nos dispositivos e redes de comunicação móveis têm possibilitado o desenvolvimento de aplicações em várias áreas como, por exemplo, transporte e logística, monitoramento ambiental e segurança. Contudo, estes aplicativos devem permitir a comunicação e coordenação entre os nós móveis, os quais podem ser pessoas, veículos ou robôs móveis. Tais nós móveis requerem tecnologias que ofereçam baixa latência e alta taxa de disseminação de dados, juntamente com estabilidade, escalabilidade e alta disponibilidade [1], [2].

O presente artigo tem por objetivo introduzir os passos básicos para o desenvolvimento de aplicações de rastreamento e comunicação utilizando o *middleware* SDDL (Scalable Data Distribution Layer), por meio do protótipo de Acompanhamento Remoto de Fiscais e Frotas - ARFF, que foi desenvolvido em apenas três meses, principalmente devido ao suporte de comunicação móvel do SDDL e seu intrínseco modelo *Publish-Subscribe* centrado em dados. Inicialmente, na Seção 2, são descritos sucintamente os conceitos e arquitetura do SDDL. A Seção 3 discute alguns trabalhos relacionados sobre *middleware* para aplicações móveis. Já a Seção 4, apresenta o ARFF, suas funcionalidades e a implementação de seus principais módulos. Por fim, a Seção 5 apresenta as considerações finais e possíveis evoluções do sistema.

2 SDDL

O SDDL (Scalable Data Distribution Layer) [3] [4] [1] é um *middleware* de comunicação que conecta nós estacionários em uma rede central (*core*) cabeada de alta velocidade com comunicação baseada na especificação *Data Distribution Service for Real-time Systems* (DDS) [5], à nós móveis com uma conexão sem fio baseada em IP (*IP-based*). O SDDL utiliza dois protocolos de comunicação: *Real-Time Publish-Subscribe* (RTPS) *Wire Protocol* [6] para a comunicação cabeada dentro da rede central do SDDL, e *Mobile Reliable* UDP (MR-UDP) *Protocol* [1] [3] para a comunicação entre os nós móveis e a rede central.

Os elementos da rede central baseiam-se no modelo DCPS (*Data-Centric Publish-Subscribe*) do DDS, onde os Tópicos DDS são definidos para serem usados para comunicação e coordenação entre tais elementos da rede central. Como parte da rede central, alguns nós do SDDL possuem funções distintas, tais como: (i) *Gateway*, (ii) *Controller* and (iii) *GroupDefiner*.

O *Gateway* (GW) define um único ponto de acesso (*Point of Attachment* – PoA) para as conexões com os nós móveis. O *Gateway* é responsável por gerenciar uma conexão MR-UDP separada para cada um dos nós móveis, encaminhando qualquer mensagem específica da aplicação ou informação de contexto para a rede central, e em direção oposta, convertendo mensagens DDS para mensagens MR-UDP e as entregando de maneira confiável aos nós móveis correspondentes.

O *Controller* é uma aplicação para a visualização da posição de todos os nós móveis (ou qualquer outra informação de contexto), além de permitir o gerenciamento de grupos e o envio de mensagens *unicast*, *groupcast* ou *broadcast* para os nós móveis.

Um nó de processamento é qualquer tipo de nó que realiza alguma operação de processamento em um dado produzido por nós móveis. Como exemplo, o SDDL tem

um nó de processamento chamado *GroupDefiner* que é responsável por avaliar as associações de grupo dos nós móveis. Para tal, o *GroupDefiner* se subscreve a um Tópico DDS onde as mensagens ou dados de contexto dos nós móveis são disseminados e então mapeia cada nó móvel em zero, um ou mais grupos, de acordo com alguma lógica de processamento de grupos específica da aplicação. Esta informação de associação de grupo é então compartilhada com todos os *Gateways* na rede central usando um Tópico DDS específico para este controle.

3 Trabalhos Relacionados

Na literatura pode ser encontrada uma grande quantidade de soluções de comunicação, entretanto, poucos demonstram como os *middlewares* podem ser utilizados na construção de aplicativos. Na prática, ocorre o desenvolvimento de estudos de caso para avaliação dos *middlewares*:

SIENA [7] é um serviço de notificação de eventos *publish/subscribe* que possui duas entidades distintas: clientes e servidores. Os servidores são interconectados (a interconexão representa o serviço de eventos) como uma rede distribuída objetivando escalabilidade, e fornecem aos clientes pontos de acesso por meio da interface do *sistema Publish/Subscribe*. Em [7], com o intuito de estudar o desempenho do SIENA em uma rede *wireless* de baixo desempenho e alta taxa de erro, foi desenvolvido um sistema de leilão ponto-a-ponto (P2P) que permitisse a compra e venda de produtos. Compradores e vendedores são vistos como componentes independentes.

Outros artigos como em [2],[8][9], explicam a arquitetura dos *middlewares* e desenvolvem aplicações para validar questões de desempenho. Contudo, explicações de como desenvolver aplicações utilizando tais middlewares são negligenciadas.

4 ARFF

O sistema de Acompanhamento Remoto de Fiscais e Frotas (ARFF) é um protótipo desenvolvido sobre a API do SDDL, cujo objetivo é gerenciar e agilizar o processo de fiscalização de ônibus, bem como demonstrar a viabilidade no desenvolvimento de aplicações que distribuam dados de contexto de forma escalável (e simples para o desenvolvedor da aplicação) para centenas de milhares de nós móveis (ônibus).

Fiscais podem estar em comandos, ou seja, em lugares pré-definidos ao longo de estradas/rodovias, onde fiscalizam veículos (ônibus) em uma *blitz*, ou podem estar em centrais de controle/monitoramento para gerenciar eventos e auxiliar outros fiscais que estão realizando as fiscalizações em campo. Desta forma, as funcionalidades principais são: (i) identificar no mapa a posição atual de cada ônibus/fiscal; (ii) Enviar mensagens em *unicast*, *groupcast* e *broadcast*; (iii) Acompanhar, a partir da central de monitoramento, as inspeções em andamento; (iv) Delegar uma inspeção a um fiscal; e (v) Cadastrar regiões que os ônibus não podem entrar e avisar quando um ônibus entrar ou sair dessa região.

Os eventos de troca de informação entre o ARFF e o SDDL são assíncronos, ou seja, a aplicação não precisa a todo o momento verificar se há novos dados de contexto a serem exibidos. Assim que uma mudança de contexto ocorre, o SDDL dispara um evento que é tratado pela aplicação. A Figura 1 mostra, em alto nível, a arquitetura do ARFF.

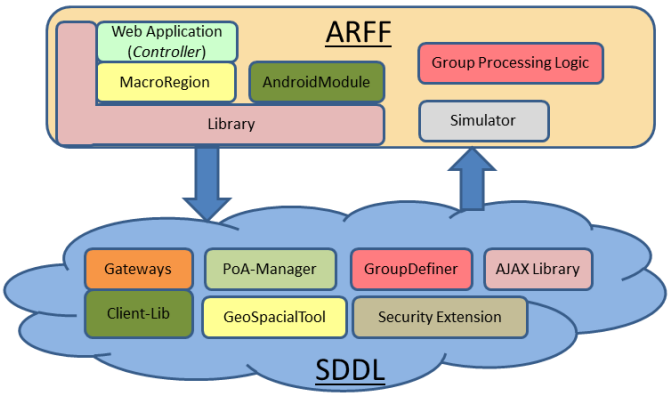

Figura 1 - Arquitetura do ARFF

As seções a seguir explicam o funcionamento de cada módulo do ARFF e como foi feita a implementação dos principais componentes usando a API do SDDL. Mais detalhes sobre o SDDL e seu funcionamento podem ser encontrados em [3] [4] [1].

4.1 Web Application

Da central de monitoramento do ARFF (*controller*) é possível saber a localização de cada ônibus e fiscal, bem como suas respectivas informações. Cada cor tem um significado, por exemplo, um ônibus em vermelho significa que o veículo foi reprovado na última inspeção. Caso haja algum tipo de conexão com a internet no momento de uma inspeção, é possível que a central acompanhe todos os procedimentos que o fiscal está realizando, já sendo informados os itens inspecionados aprovados ou reprovados. A Figura 2 exibe a tela da central de monitoramento.

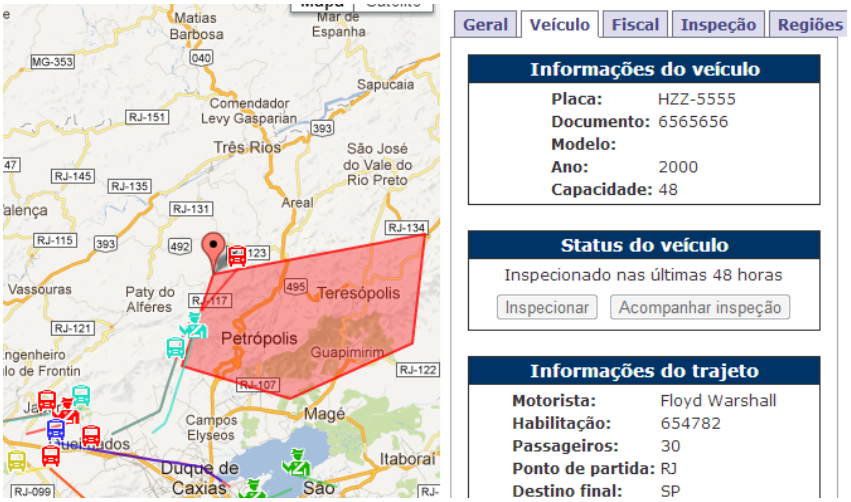

Figura 2 - Central de Monitoramento

Paralelamente ao *controller*, foi desenvolvido um módulo *android* multifuncional para o uso de fiscais durante atividades de campo. O aplicativo conta com uma janela de mapa, onde o fiscal tem acesso a sua localização geográfica. Por meio de um *chat*, o fiscal pode enviar e receber mensagens da central de monitoramento e de outros fiscais. Além disso, o aplicativo conta com uma *funcionalidade* que permite a execução de diversos tipos de inspeções e preenchimento de formulários. As informações preenchidas nos diferentes campos dos formulários são enviadas automaticamente para a central de monitoramento, o que permite o acompanhamento da inspeção durante o seu andamento. A Figura 3 mostra alguns formulários do módulo *android*.

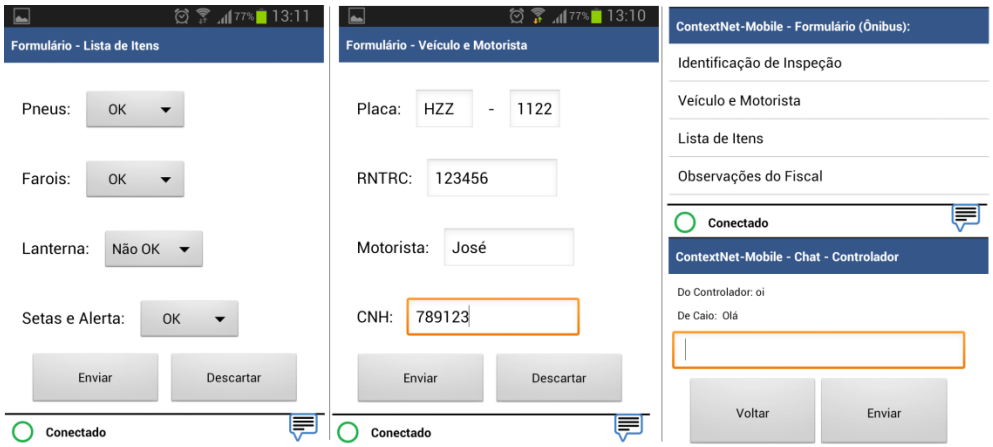

Figura 3 - Formulários do Android

Para que seja possível realizar essa troca de informações entre a aplicação web e os nós móveis, foi implementado dentro do módulo *ARFF-Library* uma classe (*HelperSerializableMessage*) responsável pelo envio de mensagens no ARRF. A Figura 4 mostra como enviar uma mensagem (i.e. a instância de um tópico) para o domínio SDDL. O parâmetro *serializableMessage* é a mensagem que será enviada, já a variável *privateMessageTopic* conterá as informações de endereçamento, ou seja, se a mensagem será endereçada a um nó móvel específico, a um grupo de nós ou para todos os nós.

```
private void WriteTopic(Serializable serializableMessage,
                        PrivateMessageTopic privateMessageTopic) throws IOException {
  ApplicationMessage applicationMessage = new ApplicationMessage();
  applicationMessage.setContentObject(serializableMessage);
  privateMessageTopic.message = Serialization.getObjectByteStream(applicationMessage);

  ddsNode.writeTopic(PrivateMessageTopic.class.getSimpleName(), privateMessageTopic);
}
```

Figura 4 - Enviando uma mensagem para um Tópico do Domínio

Para o envio de uma mensagem em *broadcast* é necessário informar ao SDDL que não há um *Gateway*, nó móvel e grupo específico de envio. Para tal é necessário atribuir o valor de *broadcast*, como visto na Figura 5. O passo seguinte é usar o método da Figura 4 para enviar a mensagem para o domínio.

```
PrivateMessageTopic privateMessageTopic = new PrivateMessageTopic();

privateMessageTopic.leastSignificantBitsGatewayId = UniversalDDSLayerFactory.BROADCAST_FLAG;
privateMessageTopic.mostSignificantBitsGatewayId = UniversalDDSLayerFactory.BROADCAST_FLAG;
privateMessageTopic.leastSignificantBitsVehicleId = UniversalDDSLayerFactory.BROADCAST_FLAG;
privateMessageTopic.mostSignificantBitsVehicleId = UniversalDDSLayerFactory.BROADCAST_FLAG;
privateMessageTopic.groupId = UniversalDDSLayerFactory.BROADCAST_FLAG;
privateMessageTopic.groupType = UniversalDDSLayerFactory.BROADCAST_FLAG;
```

Figura 5 - Mensagem em broacast

Os passos para enviar mensagens para grupos é semelhante, a diferença reside na propriedade *groupId* e *groupType* que receberão seus respectivos valores. Por questões de implementação, no ARFF foi convencionado como padrão para o *groupId* o valor um para o grupo de fiscais e dois para o grupo de ônibus. Já para o envio de mensagens para um nó móvel específico, as únicas informações necessárias são o identificador do *Gateway* e do nó móvel (ônibus). O *groupType* é utilizado na lógica de processamento de grupos separando os nós móveis por tipo. Poderiam ter sido definidos dois tipos de grupos no ARFF - um tipo para os ônibus e outro para os fiscais. Para o tipo de grupo ônibus, poderiam haver vários grupos - dos ônibus já fiscalizados ou não fiscalizados, por exemplo.

4.2 Group Process Logic

O *GroupDefiner* é responsável por avaliar as adesões de grupo de todos os nós móveis [3]. Este componente tem por objetivo definir os grupos em que cada nó móvel deve participar. Para desenvolver o serviço de processamento de grupos, deve-se implementar a interface *GroupSelector* ou *GroupSelectorSupportingProhibitedGroups* caso no sistema a ser desenvolvido haja a necessidade de restrições de regiões, ou seja, se existem regiões no mapa que os ônibus não podem circular. Ambas as interfaces possuem o método *processGroups*, onde ficará a lógica de definição dos grupos.

Para a codificação dos grupos proibidos, o primeiro passo é carregar a lista de regiões cujos ônibus não podem entrar (mais detalhes sobre as regiões podem ser vistas na seção 4.3). Para o desenvolvedor, uma região é um conjunto de pontos. Em seguida deve-se testar se o objeto é um ônibus e se está dentro de uma região proibida. A Figura 6 exibe como processar os grupos da aplicação.

```java
@Override
public Set<Integer> processGroups(Message nodeMessage) {
  Object applicationObject =
      Serialization.getObjectFromBytes(nodeMessage.content);
  Set<Integer> groupList = new HashSet<Integer>();

  if (applicationObject instanceof Inspector) {
    groupList.add(1);
  }
  else if (applicationObject instanceof Vehicle) {
    groupList.add(2);
  }
  return groupList;
}
```

```java
@Override
public Set<Integer> processProhibitedGroups(Message nodeMessage) {
  Object applicationObject =
    Serialization.getObjectFromBytes(nodeMessage.content);
  // code for load regions
  Set<Integer> groups = new HashSet<Integer>();
  if (applicationObject instanceof Vehicle) {
    Vehicle vehicle = (Vehicle) applicationObject;
    try {
      for (MacroRegion region : listRegion) {
        if (region.isVehicleInMacroRegion(vehicle)) {
          groups.add(region.getId());
          prohibitedGroups.add(region.getId());
        }
      }
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  return groups;
}
```

Figura 6 - Processamento de Grupos

Percebe-se também que o identificador da região foi adicionado a uma lista de grupos proibidos (variável de classe: *prohibitedGroups*). Isso é necessário, pois a interface *GroupSelectorSupportingProhibitedGroups* possui o método *isProhibitedGroup*, o qual é responsável por repassar para o SDDL uma lista dos grupos proibidos. Com os exemplos mostrados, nota-se que o SDDL fornece um mecanismo simples e ágil para implementar a lógica de processamento de grupo para qualquer tipo de aplicação.

4.3 Macro Region

A segurança é um requisito essencial no monitoramento/acompanhamento de frotas. Assim, se faz necessário a delimitação de regiões com um grande índice a assaltos ou acidentes, por exemplo. Logo, é importante que nenhum veículo da frota adentre nesta região, ou caso seja necessária à entrada, esse veículo pode ser acompanhado de perto pela central. A seguir será mostrado como foi feito esse tipo de monitoramento no ARFF utilizando a API do SDDL.

Na prática, para o SDDL, cada região dita como proibida é na verdade um novo grupo, e assim está ligada ao *GroupDefiner*. No ARFF, foi criado um serviço que identifica o evento de entrada/saída de um nó móvel (ônibus) em uma região proibida.

```
@Override
public void onNewData(Object topicSample) {
  if (topicSample instanceof GroupAdvertisementTopic) {
    String msg = "";
    GroupAdvertisementTopic region = ((GroupAdvertisementTopic) topicSample);
    boolean showAlert = false;
    for (Integer groupId : region.groupOperationCollection) {
      /* Code-If groupId is greater than 2, object has entered,
      else left a region - Set Message and show alert */
      UUID gatewayId = new UUID(region.mostSignificantBitsGatewayId,
                                region.leastSignificantBitsGatewayId);
      UUID vehicleId = new UUID(region.mostSignificantBitsVehicleId,
                                region.leastSignificantBitsVehicleId);
      ChangedSpecialRegion changedSpecialRegion =
        new ChangedSpecialRegion(msg, region.groupType, gatewayId,
                                 vehicleId, groupId, showAlert);
      SendMessage(changedSpecialRegion);
    }
  }
}
```

Figura 7 - Evento de Entrada/Saída de Região Proibida

Para que o serviço mostrado na Figura 7 funcione, é necessário implementar a interface *UDIDataReaderListener*. Assim, o serviço é notificado, assincronamente, sobre novos dados de entrada/saída de grupos recebidos no domínio DDS. Por questão de padronização do SDDL, sempre que há uma entrada em um grupo, o é utilizado um identificador positivo, caso contrário é utilizado um identificador negativo. Portanto, qualquer identificador de grupo maior que dois (os valores um e dois são reservados para fiscais e ônibus respectivamente) ou menor que menos dois implica na entrada/saída de um ônibus em uma região. Para que a web application (*controller*) seja notificada deste evento, é criado uma instância do objeto *ChangedSpecialRegion* com uma mensagem de alerta destinada ao motorista do ônibus, tipo de grupo, os identificadores do gateway e do ônibus, e *boolean* (*showAlert*) informando se algum alerta deverá ser emitido no mapa. Esse objeto é enviado para o domínio DDS

4.4 Simulator

O módulo de simulação do ARFF foi desenvolvido com o intuito de emular vários nós móveis e consequentemente o envio/recebimento de informações de contexto (latitude, longitude, velocidade e a hora de ocorrência do evento). Para a implementação foi utilizada a *Client-Lib* do SDDL, que é um componente de software para implementar os aplicativos nos clientes móveis. Ela esconde a maior parte dos detalhes dos protocolos de comunicação e trata problemas de conectividade com os *Gateways* [3].

Para desenvolver o simulador, é preciso implementar a interface *NodeConnectionListener* do MR-UDP, pois ela representa os eventos (ex.: conexão, desconexão, chegada de novas mensagens) entre o nó móvel e o Gateway. Toda abstração de envio da mensagem fica encapsulada no *ApplicationMessage*, e basta que o objeto com suas novas informações de contexto e seu respectivo identificador sejam informados. De maneira simples, consegue-se emular um grande número de nós móveis sem sobrecarregar a capacidade de processamento do servidor. A Figura 8 mostra a utilização da *Client-Lib* do SDDL.

```
@Override
public void run() {
  // Some code ...
  Point2D.Double newPos = new Point2D.Double();
  // code for calculate new positions
  long time = System.currentTimeMillis();

  InspectorTrackingInformation tracking =
    new InspectorTrackingInformation(newPos.getX(), newPos.getY(),
                                     10.1, time - (30 * 1000));
  inspectorLocalInfo.getTrackingInformation().add(tracking);
  lastPos = newPos;

  applicationMessage = new ApplicationMessage();
  applicationMessage.setContentObject(inspectorLocalInfo);
  applicationMessage.setTagList(tags);
  applicationMessage.setSenderID(inspectorLocalInfo.getId());

  try {
    myConnection.sendMessage(applicationMessage);
  }
  catch (IOException e) {
    e.printStackTrace();
  }
}
```

Figura 8 - Envio de dados de contexto

4.5 Demonstração no SBRC

A demonstração pode ser feita com o uso dois notebooks (com acesso a internet), um para emular os nós móveis e outro para emular o núcleo SDDL e rodar a aplicação web, além de um *tablet* com o Sistema Operacional Android. A demonstração inclui a emulação de 10 ônibus e 5 fiscais, sendo que um dos fiscais deverá utilizar o *tablet*, para demonstrar as funcionalidades do sistema. Os nós móveis (fiscais e ônibus) possuirão localização arbitrária e atualizações a cada 15 segundos. Em paralelo, uma demonstração com 5000 nós móveis pode ser realizada para mostrar a escalabilidade e desempenho, apesar do grande volume de dados.

5 Conclusão

Este artigo apresenta o ARFF, desenvolvido com o uso do SDDL (*Scalable Data Distribution Layer*), um *middleware* de comunicação distribuída. Com o ARFF, foi possível mostrar como implementar o envio de mensagens e de dados de contexto, assim como mostrar como é realizada a definição de grupos e como foi definido o serviço de descoberta de entrada/saída de regiões proibidas. A versão corrente, bem como sua documentação, estão disponíveis em: http://www.lac.inf.puc-rio.br/arff/.

Como trabalhos futuros, serão desenvolvidas APIs de *QoS*, segurança e balanceamento de carga. Outras evoluções poderiam ser realizadas através do uso de informações contexto e prover mecanismos que permitissem a associação de regras de contexto com outras funcionalidades de colaboração. Por exemplo, caso esteja chovendo e um veículo transportando carga perigosa, uma mensagem automática seria enviada informando que uma determinada velocidade máxima não deverá ser ultrapassada.

Referências

[1] L. David, R. Vasconcelos, L. Alves, R. André, G. Baptista, and M. Endler, “A Large-scale Communication Middleware for Fleet Tracking and Management,” in Simposio Brasileiro de Redes de Computadores e Sistemas Distribuidos (SBRC 2012), Salao de Ferramentas, 2012, pp. 964–971.

[2] A. Corradi, L. Foschini, and L. Nardelli, "A DDS-compliant infrastructure for fault-tolerant and scalable data dissemination," in The IEEE symposium on Computers and Communications, 2010, pp. 489–495.

[3] L. David, R. Vasconcelos, L. Alves, R. André, G. Baptista, and M. Endler, "A Communication Middleware for Scalable Real-time Mobile Collaboration," in Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2012 IEEE 21st International Workshop on, 2012, pp. 54–59.

[4] R. O. Vasconcelos, L. David, L. Alves, R. André, and M. Endler, "Real-time Group Management and Communication for Large-scale Pervasive Applications." Monografias em Ciência da Computação - MCC 05/2012, Dep. de Informática, PUC-Rio, ISSN 0103-9741, Rio de Janeiro, 2012.

[5] OMG, "Data-Distribution Service for Real-Time Systems (DDS)." 2006.

[6] OMG, "The Real-time Publish-Subscribe (RTPS) Wire Protocol DDS Interoperability Wire Protocol Specification (DDS-RTPS)," 2010. [Online]. Available: http://www.omg.org/spec/DDS-RTPS/2.1/. [Accessed: 25-Feb-2013].

[7] M. Caporuscio, P. Inverardi, and P. Pelliccione, "Formal Analysis of Clients Mobility in the Siena Publish / Subscribe Middleware," Technical report, Department of Computer Science and University of L'Aquila, 2002.

[8] A. Corradi and L. Foschini, "A DDS-compliant P2P infrastructure for reliable and QoS-enabled data dissemination," 2009 IEEE International Symposium on Parallel & Distributed Processing, pp. 1–8, May 2009.

[9] A. Rowstron and P. Druschel, "Pastry : Scalable , decentralized object location and routing for large-scale peer-to-peer systems," in Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, 2001, no. November 2001, pp. 329–350. Disponível em: http://www.inf.puc-rio.br/~bib-di/mccedit.html. Acesso em: 14 ago. 2002.