Tesis defendida por

Carlos Fernando Caloca De la Parra

y aprobada por el siguiente Comité

Dr. José Antonio García Macías
Director del Comité

Dr. Carlos Alberto Brizuela Rodríguez
Miembro del Comité

Dr. Rolando Menchaca Méndez
Miembro del Comité

Dr. Andrey Chernykh
Miembro del Comité

Dr. José Antonio García Macías
Coordinador
del Posgrado en Ciencias de la Computación

Dr. Jesús Favela Vara
Director de la
Dirección de Estudios de Posgrado

Noviembre 2013

CENTRO DE INVESTIGACIÓN CIENTÍFICA Y DE EDUCACIÓN SUPERIOR DE ENSENADA, BAJA CALIFORNIA

Programa de Posgrado en Ciencias
en Ciencias de la Computación

Middleware de adaptación de protocolos de capa de red basada en información de contexto, enfocada a redes vehiculares

Tesis
para cubrir parcialmente los requisitos necesarios para obtener el grado de

Doctor en Ciencias
Presenta:

Carlos Fernando Caloca De la Parra

Ensenada, Baja California, México
2013

Resumen de la tesis de **Carlos Fernando Caloca De la Parra,** presentada como requisito parcial para la obtención del grado de Doctor en Ciencias en Ciencias de la Computación.

Middleware de adaptación de protocolos de capa de red basada en información de contexto, enfocada a redes vehiculares

Resumen aprobado por:

_____
Dr. José Antonio García Macías

Los avances en los sistemas de cómputo de los nuevos automóviles han motivado la comunicación entre vehículos y un nuevo subtipo de redes inalámbricas denominadas "Vehicular Ad Hoc Networks" (VANETs). Las redes VANET representan un gran reto para los protocolos de capa de red debido a sus características dinámicas y a las diferentes necesidades de las aplicaciones vehiculares. Debido a lo anterior, en este trabajo se propone el diseño de una arquitectura middleware (PLUGAM) que permite adaptar el comportamiento de la capa de red del sistema vehicular, utilizando información del contexto local. La arquitectura PLUGAM es genérica, es decir está diseñada para crear una capa de red adaptativa con elementos de contexto, protocolos de red, algoritmos de adaptación y aplicaciones vehiculares definidos por el usuario en la forma de componentes. Estos componentes son introducidos al middleware utilizando la tecnología plug-ins. Además, debido a que las necesidades de las aplicaciones vehiculares no pueden ser atendidas por un solo tipo de protocolo de capa de red, esta arquitectura promueve la coexistencia de múltiples protocolos de red en el sistema vehicular.

Se presentaron dos casos de estudio de protocolos adaptativos, los cuales son construidos con la arquitectura del middleware de adaptación, cada uno trata uno de los dos tipos de adaptaciones exploradas en este trabajo. La evaluación de este trabajo consistió en un análisis de costo/desempeño al prototipo, basada en un conjunto de mediciones sobre el costo de procesamiento y de memoria adicional. Además se presenta una simulación de uno de los casos de estudio con el fin de analizar el impacto que tiene la adaptación propuesta aquí. De la evaluación se puede concluir que implementar la arquitectura propuesta resulta viable, los requerimientos de memoria de middleware de adaptación son moderados, se podría implementar en sistemas vehiculares o dispositivos móviles actuales. El tiempo de procesamiento introducido por el middleware en cualquiera de los modos de adaptación es razonable. Debido a que la arquitectura PLUGAM es un sistema distribuido completamente descentralizado, es escalable en cuanto al número de nodos en la red. Por último, el uso de tecnología plug-in resulta factible y una buena opción para la construcción de sistemas vehiculares, en especial si se utiliza la tecnología OSGi e iPOJO ya que genera un costo de procesamiento muy pequeño a costa de elevar el tiempo de inicialización y costo en memoria.

Palabras clave**: Middleware, Redes Inalámbricas Vehiculares, Adaptación basada en contexto local**

Abstract of the thesis presented by **Carlos Fernando Caloca De la Parra** as a partial requirement to obtain the Doctor in Science degree in Computer Science.

Adaptation Middleware for Network Layer Protocols in VANETS Based on Context Information

Abstract approved by:

_____

Dr. José Antonio García Macías

Advances in the computer systems of today's automobiles have motivated the use of new forms of communication between vehicles and a new type of wireless mobile network called "Vehicular Ad Hoc Networks" (VANETs). These networks pose a great challenge for network layer protocols due to their dynamic characteristics and to the different needs of vehicular applications. This thesis work proposed the design of a middleware architecture (PLUGAM) that allows the adaptation of the network layer behavior in the vehicular system, using local context information. PLUGAM is a generic architecture, as it is designed to create an adaptive network layer using different context elements, network protocols, adaptation algorithms and vehicular applications defined by the user in the form of components. These components are introduced into the middleware using plug-in technology. Furthermore, because the needs of vehicular applications cannot be met by a single type of network layer protocol, this architecture promotes the coexistence of multiple network protocols in the vehicular system.

Two case studies of adaptive protocols built with the middleware architecture were presented, each representing one of the two key adaptations explored in this work. The evaluation of this work consisted of a cost/performance analysis, based on a set of processing and memory cost measurements done to the prototype. Additionally, a simulation of one of the case studies is presented, in order to analyze the impact of the proposed adaptation. From the evaluation we can conclude that implementing the proposed architecture is feasible, memory requirements of the adaptation middleware are moderate and it can be implemented in near future vehicular systems and even in today's mobile devices hardware. The processing cost, measured in time, introduced by the middleware in any of the adaptation modes is reasonable by communication systems standards. Since the PLUGAM architecture is a completely decentralized distributed system, it is scalable in the number of nodes in the network; in regard to the number of network layer protocols coexisting in the vehicular system, scalability is affected mainly in terms of the memory cost generated by each protocol. Furthermore, the use of plug-in technology for building distributed systems such as a vehicular system is feasible and a good option, especially using OSGi and iPOJO technologies since they generate a very small processing cost, however at the expense of increasing the initialization time and memory cost.

Keywords: **Middleware, Wireless Vehicular Ad-Hoc Networks, Adaptation Based On Local Context.**

Para ti, mi mejor amiga, mi compañera de vida, que has estado siempre presente brindando apoyo en todos los aspectos. Gracias por estar siempre ahí, te amo.

Agradecimientos

Al Dr. José Antonio García Macías, por su amistad, su apoyo, sus conocimientos, su experiencia, su paciencia y su motivación que ayudaron a que terminara mis estudios con éxito.

A mi comité de tesis, los doctores Carlos Brizuela, Andrey Chernykh y Rolando Menchaca, por su tiempo, sus consejos, sus correcciones y sus invaluables comentarios que me ayudaron a concluir mi trabajo.

Al Centro de Investigación Científica y de Educación Superior de Ensenada (CICESE), forjador de mi formación académica y personal. Asimismo, agradezco a todos los empleados de CICESE por su atención y amabilidad en todo lo referente al desarrollo de mis estudios doctorales.

Al Consejo Nacional de Ciencia y Tecnología (CONACyT), por brindarme los medios económicos para cursar este posgrado.

A mi esposa Lili, por su interminable apoyo y comprensión a lo largo de estos años.

A mi familia, por toda la confianza, estímulo y apoyo invaluable que me ha ayudado a seguir adelante.

Contenido

Página

Contenido

Contenido

Página

Lista de Figuras

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1 Antecedentes

En décadas recientes, las personas alrededor del mundo están integrando cada vez más en sus vidas a las computadoras y a la Internet. Particularmente, las redes inalámbricas han tenido un desarrollo explosivo, en ámbitos tanto civil como militar. Una de las principales razones de éxito de estas redes es la posibilidad de utilizar algunos de los servicios provistos por la red, aún cuando se está en movimiento.

Las redes inalámbricas se han integrado a una gran variedad de dispositivos móviles tales como computadoras portátiles, teléfonos inteligentes, dispositivos de sensado, y recientemente en automóviles.

Los avances en los sistemas de cómputo de los nuevos automóviles y la proliferación de tecnologías de redes inalámbricas, han hecho que los vehículos ya no sean vistos como simples medios de transporte. Actualmente, los consumidores esperan que sus vehículos les provean de información y servicios de comunicación; lo anterior ha motivado la introducción de comunicación entre vehículos, o también llamado *"Inter-Vehicle Communications"* Reichardt *et al.* (2002).

La tecnología inalámbrica permite que el vehículo capte más información de su entorno, inclusive fuera de su rango de comunicación con ayuda de otros vehículos. El captar información más lejana ayuda a prevenir las acciones del conductor y mejorar la seguridad en el camino. Los automóviles están cambiando de ser sistemas autónomos y aislados, hacia nodos cooperativos formando una red sobre ruedas e intercambiando información sobre su estado, comportamiento y su entorno. Se ha formado una co-

munidad entera alrededor de la problemática de las comunicaciones vehiculares y en particular las VANETs o *"Vehicular Ad Hoc Networks"* (Lind *et al.* (1998)).

Los entornos de comunicación vehicular son caracterizados por vehículos de alta movilidad, cambios frecuentes en la topología, y una gran variación en el número de vehículos en cierta región (Schoch *et al.* (2008); Adler *et al.* (2006)), entre otras características dinámicas de la red.

Las aplicaciones vehiculares que en un futuro operarán en conjunto con VANET's, van desde simples intercambios de información describiendo el estado del vehículo, hasta una compleja administración de tráfico a gran escala. Se han propuesto una gran diversidad de aplicaciones potenciales, las cuales necesitan diferentes requerimientos de comunicación.

La mayor parte del esfuerzo de investigación en el área de VANETs se centra en los aspectos de capa de red tales como el enrutamiento, esto debido a los retos de las comunicaciones inalámbricas y la movilidad de los vehículos. Un protocolo de capa de red define las reglas y estándares para transportar datos de la fuente de información hacia el destino, atravesando varios nodos intermedios. Dentro de los protocolos de capa de red más conocidos se encuentran los de diseminación/broadcast y enrutamiento unicast.

1.2 Planteamiento del problema

Las características dinámicas de las VANETs, tales como la movilidad de vehículo, la condición de canal inalámbrico, la densidad de nodos, etc., y los requerimientos de comunicación de las aplicaciones pueden influir en el protocolo de capa de red, por lo que existe la necesidad de que estos protocolos tomen en cuenta a este contexto para adaptar su comportamiento al dinamismo de la red y las aplicaciones.

Existen propuestas de protocolos de red VANET adaptativos que abordan parte de este dinamismo. Por ejemplo la propuesta de Adler *et al.* (2006) que adaptan el área y tiempo de diseminación, Zhao y Cao (2006) cuyo trabajo se centra en adaptar el protocolo cambiando el algoritmo de enrutamiento o calibrando sus parámetros de acuerdo a información del contexto, Eichler *et al.* (2006) que adaptan la prioridad del envío de mensajes entre nodos en base al beneficio potencial del mensaje, entre otros.

Las propuestas de protocolos adaptativos mencionadas anteriormente ofrecen una solución parcial al problema, debido a que el tema del dinamismo en redes VANETs es muy complejo. Ademas se encontró que estas propuestas, así como muchas otras, están optimizadas para escenarios específicos o definidos para un protocolo de red en particular. Otro inconveniente es que el algoritmo de adaptación está fuertemente integrado con la funcionalidad del protocolo de red, por lo que resulta difícil extraer y reutilizar la lógica adaptativa en otros protocolos.

Es posible generalizar las propuestas de protocolos adaptativos, proponiendo arquitecturas para construir sistemas tipo *"frameworks"* o *"middlewares"*, orientados a proporcionar las propiedades de adaptación u ofrecen un marco de trabajo para realizar soluciones de adaptación.

Un *middleware* es un sistema que provee una solución estándar y re-utilizable para tareas de programación común requeridas por la aplicación, tales como almacenamiento persistente, control de concurrencia, diversidad de sistema operativos y pila de protocolos, a diferencia de un *framework* en el cual se define la estructura de la aplicación. Por lo general, los *middleware* se ubican entre las capa de aplicación y la capa de red del modelo de interconexión de sistemas abiertos OSI (Day y Zimmermann (1983)).

Entre las propuestas de arquitecturas genéricas descritas anteriormente, existen

aquellas que se enfocan en adaptar a las aplicaciones en vez de a los protocolos de capa de red (Capra *et al.* (2003); Popovici *et al.* (2011)), otras adaptan protocolos de capa de red pero no toman en cuenta un escenario de múltiples protocolos dentro la capa de red (Ramdhany *et al.* (2009)), algunos otros proponen cómo modelar y proveer el contexto pero no cómo realizar la adaptación utilizando la información generada a partir de éste (Wang *et al.* (2004); Salber *et al.* (1999)), o al revés, que proponen un marco de trabajo o modelo para adaptar, pero no definen como introducir la información de contexto al algoritmo de adaptación (Liu y Cheng (2008); Popovici *et al.* (2003)).

Por lo anterior, se propone el siguiente objetivo de tesis.

1.2.1 Objetivo general

Diseñar una arquitectura *middleware* que permita crear sistemas vehiculares con una capa de red adaptativa, con base en la información de contexto del vehículo.

1.2.2 Objetivos específicos

1. Desarrollar una arquitectura genérica que disminuya la complejidad de diseñar y construir una capa de red adaptativa.

2. Diseñar la arquitectura de tal forma que permita construir una amplia variedad de propuestas de protocolos adaptativos que existen en la literatura.

3. Proveer a los protocolos de capa de red la habilidad de modificar su comportamiento en base a la información de contexto.

4. Establecer la manera de definir y proveer al *middleware* los contextos que serán utilizados para realizar las adaptaciones.

5. Determinar la viabilidad práctica de la arquitectura propuesta en este trabajo.

1.3 Metodología

El desarrollo de este trabajo de tesis se realizó de la siguiente manera:

- *Etapa 1.* Revisión de la literatura y proyectos existentes. Se revisó el estado del arte en redes inalámbricas de vehículos (VANET), soluciones de adaptación, arquitecturas *middleware* y protocolos adaptativos en MANET y VANET. Como resultado de la revisión del estado del arte se encontraron las siguientes áreas de oportunidad: proponer una solución genérica para proporcionar adaptación en base a contextos a los protocolos de red y aplicaciones vehiculares, tener un sistema vehicular que funcione con varios tipos de protocolos de red que satisfagan las necesidad de comunicación de las aplicaciones vehiculares.
- *Etapa 2.* Análisis y diseño de la arquitectura del *middleware*: Se realizó el diseño de la arquitectura en dos fases, en la primera se propuso un diseño general en base a los requerimientos obtenidos en la literatura. En ésta se abordaron los siguientes temas:

  – Buscar alternativas para proveer a la arquitectura de componentes que se desarrollen de manera independiente, los cuales se agreguen a ésta para proporcionar la funcionalidad requerida. Se optó por la tecnología *plug-ins*.

  – Se propuso un modelo de adaptación, que relaciona al algoritmo que realiza la adaptación, con la información de contexto que sirve como entrada, mediante el cual se obtienen salidas que utilizan los protocolos de red para modificar su comportamiento.

  – Se propuso cómo declarar adaptaciones en la arquitectura.

  En la segunda fase del diseño, a medida que se construía un prototipo, se obtuvo el diseño final de la arquitectura. En esta fase se abordaron los siguientes temas:

  – Se definieron las interfaces para cada tipo de *plug-in*, mediante las cuales el *middleware* interactua con los componentes que estos ofrecen.

  - Se estableció la interacción entre las aplicaciones vehiculares, protocolos de red y *middleware*.

- *Etapa 3.* Definición de casos de estudio: Con motivo de ejemplificar la funcionalidad y uso de la arquitectura, se definieron dos casos de estudios, éstos presentan los dos tipos de adaptación que se busca construir con la misma. Para cada caso de estudio se realizó lo siguiente:

  - Se describió a detalle el escenario del caso de estudio.
  - Se detalló como se implementaría éste en la arquitectura PLUGAM.
  - Se definió como evaluar mediante simulación el caso de estudio.

- *Etapa 4.* Implementación del prototipo de *middleware*: Se desarrolló un prototipo del *middleware* de adaptación, y algunos plug-ins para proveer al *middleware* de la funcionalidad necesaria para implementar uno de los casos de estudio. El prototipo ayudó a determinar la viabilidad de la arquitectura propuesta. En esta etapa se realizó lo siguiente:

  - Se configuró una plataforma de *plug-ins* para adecuarla a las necesidades de la arquitectura.
  - Se implementó un *plug-in* que ofrece elementos de contextos obtenidos por un GPS, y otro *plug-in* que encapsula una implementación del protocolo de red OLSR.
  - Se implementó el funcionamiento del *middleware*, primeramente la detección de los componentes ofrecidos por los *plug-ins* y enseguida el motor de ejecución de las adaptaciones.
  - Se implementó el funcionamiento de los distintos modos de adaptación, además de la interacción entre las aplicaciones, protocolos de red y *middleware*.

- *Etapa 5.* Evaluación y análisis de resultados: En esta etapa se presenta una evaluación de costo/desempeño, la cual está basada en un conjunto de mediciones sobre el costo de procesamiento y de memoria adicional hechas con respecto a un prototipo en el lenguaje de programación Java (descrito en el apéndice A). Además, se presenta una simulación en Ns-2 de uno de los casos de estudio descritos en el capítulo 6, esto con el fin de analizar el impacto que tiene la adaptación propuesta en el caso de estudio e ir más allá de solo lo teórico. Por último se discute la escalabilidad de la arquitectura propuesta, además se describen los costos adicionales introducidos por el enfoque genérico de la arquitectura en contraste con los protocolos adaptativos existentes, los cuales son soluciones particulares.

1.4 Contenido de la tesis

Esta tesis contiene ocho capítulos y un apéndice los cuales se describen brevemente a continuación.

En el capítulo 2 se presenta el concepto de las redes VANETs y los modos de comunicación para aplicaciones vehiculares. Se describen los tipos de protocolos de capa de red que existen actualmente y la complejidad de crear protocolos de capa de red adaptativos para redes VANETs.

El capítulo 3 define los protocolos de capa de red adaptativos, además presenta el análisis comparativo de los trabajos encontrados en la literatura sobre estos protocolos en redes VANETs y MANETs, y los problemas de las propuestas actuales de protocolos adaptativos. El capítulo también propone el desarrollo de una arquitectura *middleware* que brinde mecanismos de adaptación a los protocolos de capa de red de un sistema vehicular, e introduce los trabajos realizados sobre arquitecturas *middleware* que proporcionan propiedades de adaptación y que proveen contexto a las aplicaciones en ambientes móviles.

En el capítulo 4 se presenta el modelo de adaptación que sustenta a la arquitectura del *middleware* de adaptación propuesta en este trabajo de tesis y describe los elementos que la componen. Introduce el concepto de *"plug-ins"*, y describe la interacción de las aplicaciones vehiculares con el *middleware*.

El capítulo 5 describe los elementos que permiten extender la funcionalidad de la arquitectura del *middleware* de adaptación, y presenta algunos sistemas vehiculares desarrollados por fabricantes de automóviles.

En el capítulo 6 se presentan dos casos de estudio realizados para mostrar el funcionamiento de la arquitectura del *middleware* de adaptación.

En el capítulo 7 se describen la evaluación del prototipo del *middleware* de adaptación, se explica en qué consistió cada experimento de evaluación y cuáles fueron los resultados obtenidos en cada uno.

El capítulo 8 contiene las conclusiones de la investigación, aportaciones, limitaciones y propuestas de trabajo futuro.

En el apéndice A se describe los detalles técnicos del prototipo desarrollado para la arquitectura propuesta en este trabajo.

Capítulo 2

Redes VANET y sus protocolos de capa de red

En este capítulo hablaremos del estado de arte en redes VANET, de sus aplicaciones potenciales y sus retos. Finalizaremos hablando de los protocolos de red que se han propuesto en la literatura VANET y describiendo los retos y complicaciones que existen en estos protocolos.

2.1 Redes VANET

Las redes inalámbricas de vehículos, llamadas en inglés *"Vehicular Ad-hoc Netwotk"* (VANET), son una subclase particular de una red móvil (*"Mobile Ad-Hoc Network"* o MANET), en la cual los vehículos son equipados con interfaces de radio comunicación de corto alcance y habilitados para comunicarse con nodos de una infraestructura de red fija y ya establecida (en caso de existir) o con otros vehículos de manera oportunista. De manera más simple, una VANET (ver figura 1) es una instancia de una red MANET la cual establece conexiones inalámbricas entre vehículos y nodos de infraestructura.

Los entornos de comunicación vehicular son caracterizados por vehículos de alta movilidad, cambios extremadamente frecuentes en la topología, y una gran variación en el número de vehículos en cierta región (Schoch *et al.* (2008); Adler *et al.* (2006)). Un nodo en una VANET debe estar compuesto de al menos un radio transmisor, sensores internos y un módulo de procesamiento, opcionalmente con uno de localización como un GPS y acceso a mapas digitales de las carreteras. En años recientes ha habido un trabajo intenso de investigación en torno a VANETs debido a la amplia variedad de servicios y aplicaciones potenciales que pueden proveer.

En la literatura se distinguen dos modos de comunicación para aplicaciones vehi-

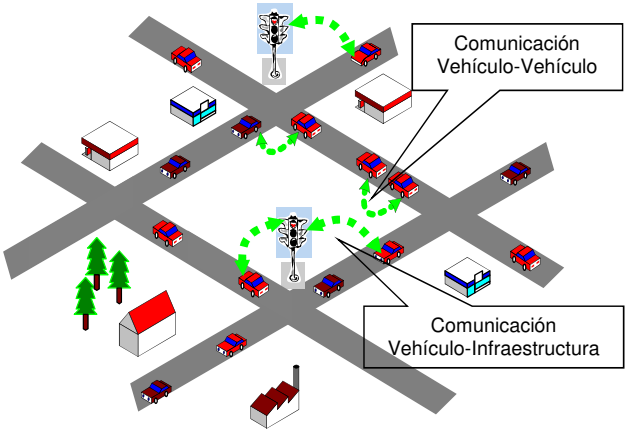

Figura 1. Redes inalámbricas vehiculares

culares. El primero es la comunicación vehículo-a-vehículo (V2V), la cual se basa en el intercambio de mensajes de manera espontánea entre los vehículos. El segundo es la comunicación vehículo-a-infraestructura (V2I), la cual utiliza una infraestructura de comunicación para entregar la información a los vehículos o para comunicar información a una red ya establecida.

En un inicio los trabajos sobre comunicación vehicular se basan en V2V, sin embargo, trabajos recientes han resaltando la importancia de comunicaciones V2I como medio de conectividad en redes con condiciones de escaso número de vehículos, además para proveer algunos servicios que no son fáciles de atender por comunicación V2V pura; por ejemplo el acceso a Internet en el camino. Inclusive el uso de comunicación V2I puede ayudar a mejorar el desempeño de aplicaciones vehiculares en etapas tempranas de la vida de una VANET (Lochert *et al.* (2007)), en donde existirían pocos vehiculos con el transmisor inalámbrico.

Los principales fabricantes de automóviles y dispositivos inalámbricos, institutos de investigación, agencias públicas y empresas privadas, están realizando investigaciones en varios tópicos relacionados a comunicaciones vehículo-a-vehículo (V2V) y vehículo-infraestructura (V2I), tales como modelación del canal inalámbrico (Taliwal

et al. (2004); Yin *et al.* (2006)), modelación de la movilidad (Lin *et al.* (2004); Bai *et al.* (2003)), algoritmos de capa de red (Lochert *et al.* (2005); Korkmaz *et al.* (2004); LeBrun *et al.* (2005)), seguridad (Leinmüller *et al.* (2006); Qian y Moayeri (2008)), estrategias de penetración del mercado (Kosch (2005); Shladover y Tan (2006)) y muchos otros temas.

En cuanto a la tecnología inalámbrica utilizada por los vehículos en una VANET, las comunidades científica y académica han favorecido el uso de tecnologías inalámbricas de área local (LAN) como base de comunicación para las aplicaciones vehiculares. Los radios IEEE 802.11b fueron utilizados para propósitos de prototipos rápidos y demostrativos. El estándar IEEE 802.11a fue seleccionado por la *"American Society for Testing and Materials"* como la base para el futuro estándar IEEE 802.11p (2010), el cual será específico para comunicaciones entre vehículos (Jiang *et al.* (2006)). En cuanto a tecnologías inalámbricas para comunicación V2I, se ha considerado el uso de WiMAX (2001), LTE (2007) y tecnologías de 3G (Garber (2002)). Un escenario ideal sería utilizar una tecnología inalámbrica de corto alcance (LAN inalámbrico) para aplicaciones con comunicación V2V o de corto alcance y para comunicación V2I utilizar tecnología 3G.

Aunque el tema de VANET y de comunicaciones entre vehículos puede ser considerado como un tema reciente, este ha obtenido mucho interés por parte de los gobiernos de países desarrollados, la academia científica y la industria automotriz, este interés se debe a la amplia variedad de servicios y aplicaciones potenciales que pueden proveer. Por parte del gobierno, la intención principal es utilizar la comunicación entre vehículos para reducir el número de accidentes en el camino (este número se ha mantenido alto aún con la introducción de cinturones de seguridad y bolsas de aire) y disminuir el uso de combustible en la red de transporte por medio de administración inteligente del tráfico vehicular. Por parte de la academia científica, lo ven como una instancia interesante de redes móviles, con nuevos retos y problemáticas a solucionar, en especial en cuanto a nuevos algoritmos de enrutamiento y de diseminación. Por parte de los fabricantes

de automóviles, la intención es darle mayor valor agregado a los nuevos automóviles al incorporarles nuevos servicios.

En Estados Unidos, Japón y Europa se han realizado una serie de programas y proyectos involucrando al concepto de Intelligent Transportation System (2005) (ITS). El concepto de ITS es un poco más general al de una VANET, en él se relacionan elementos adicionales a los vehículos, como carreteras, semáforos, señales de tráfico, etc. Además de relacionar otros tipos de vehículos diferentes al automóvil, como por ejemplo trenes, camiones de pasajeros. Entre los proyectos encaminados a desarrollar esta tecnología, en unión con el gobierno, academia e industria, tenemos a NOW (Festag *et al.* (2008)), CarTALK2000 (2001), VICS (1990), CVIS (2006), COOPERS (2006), el Car 2 Car Communication Consortium (2007), entre otros; cabe resaltar que la mayoría de estos son europeos.

Las aplicaciones vehiculares que en un futuro operarán en conjunto con VANET's, van desde simples intercambios de información describiendo el estado del vehículo, hasta una compleja administración de tráfico a grande escala. Debido a la gran diversidad de aplicaciones potenciales, varios autores han propuesto diferentes clasificaciones. Desde la perspectiva del valor o beneficio del consumidor, Abdalla *et al.* (2007) y Yousefi *et al.* (2006) consideran dos tipos: aplicaciones de seguridad en el camino y aplicaciones comerciales o de confort. Por otro lado los autores del proyecto NoW (Festag *et al.* (2008)) clasifican las aplicaciones vehiculares en tres tipos: seguridad en el camino, eficiencia de tráfico y aplicaciones de infoentrenimiento. Por último, Schoch *et al.* (2008) proponen una clasificación más detallada que consta de cuatro tipos, en base a esta se mencionarán algunos ejemplos de aplicaciones de VANETs::

- **Seguridad activa en el camino (*"Active safety"*):** En esta clasificación entran aquellas aplicaciones que muestran al conductor anuncios de advertencia sobre el camino (bajar velocidad en curva, señales de alto, frenado de emergen-

cia, anuncio de choque de vehículos), tráfico anormal (cambio de carril, zona de trabajo), servicio de SOS a centro de atención, etc.

- **Servicio público:** Por ejemplo, aplicaciones que muestran anuncios sobre el acercamiento de vehículos de emergencia o un servicio de localización de vehículos robados.
- **Mejora en la conducción del vehículo:** Por ejemplo, aplicaciones de control cooperativo de manejo automático, guia del camino y navegación mejorada, búsqueda de lugar de estacionamiento, etc.
- **Negocios móviles e infoentretenimiento:** Por ejemplo, aplicaciones de diagnóstico de automóvil por medio inalámbrico, actualizaciones de software por el aire, proveedor de servicio de Internet, notificaciones de puntos de interés, manejo de caravana, localización de vehículos, etc.

Muchos otros ejemplos de aplicaciones vehiculares pueden ser encontrados en Abdalla *et al.* (2007), en Schoch *et al.* (2008) y en la página web del proyecto Intelligent Transportation System (2005).

El desarrollo de aplicaciones de seguridad en el camino son importantes ya que cuentan con el soporte del financiamiento gubernamental, tal fue el caso de los proyectos NoW (Festag *et al.* (2008)), Fleetnet (Enkelmann (2003)) y CarTALK2000 (2001). Además, justifican la asignación de parte del espectro radioeléctrico dedicado a comunicaciones inalámbricas vehiculares (por ejemplo el IEEE 802.11p). Por otro lado, las aplicaciones que no son de seguridad en el camino son también importantes como parte del servicio ofrecido por estas redes, ya que estas pueden ofrecer un beneficio más directo e instantáneo al conductor, lo cual alentará al conductor a que invierta en el equipo de cómputo para convertir a su auto en un nodo VANET. Otro beneficio sería un estímulo a los fabricantes de automóviles a participar en la creación de esta red y sacarle provecho proponiendo sus propias aplicaciones.

A pesar de que existe más de una década de investigación en redes VANET, aún no se han implementado en el mundo real debido al tiempo que está tomando la estandarización de tecnologías inalámbricas para comunicaciones entre vehículos. En años recientes (2011 a la fecha), los fabricantes de automóviles comenzaron a introducir sistemas de cómputo más complejos y con capacidades de comunicación. La industria automotriz ha decidido llamar a estos sistemas de cómputo con el nombre de *"in-car technology"* o *"infotainment systems"*. El enfoque de estos sistemas ha sido más hacia proveer aplicaciones del tipo de infoentretenimiento (véase figura 2), como acceso a Internet, mostrar puntos de interés, obtener información del tráfico de un servidor centralizado, radio por Internet, servicios conectando el sistema del vehículo con dispositivos móviles.

Figura 2. Ejemplo de un sistema de infoentretenimiento (Chevi MyLink)

La comunicación utilizada para este tipo de aplicaciones vehiculares ha sido solamente del tipo V2I, gracias a un transmisor 3G integrado al sistema, o conectando este a un dispositivo móvil con conexión de datos 3G. Como primer ejemplo de estos sistemas de infoentrenimiento tenemos a SYNC (2007) de Ford, el cual fue pionero en el 2007. El gran éxito de SYNC ha ocasionado que a partir de 2011 otros fabricantes hayan creado sus propios sistemas, entre los cuales tenemos a Toyota Entune (2011),

Renault R-Link (2011), Chrysler Uconnect (2013) y Chevi MyLink (2012).

2.2 Estado del arte de protocolos de capa de red en VANETs

La mayor parte del esfuerzo de investigación en el área de VANETs se centra en los aspectos de capa de red tales como el enrutamiento, esto debido a los retos de la comunicaciones inalámbricas y la movilidad de los vehículos. Dentro de los protocolos de capa de red existen los de diseminación y *"broadcast"*, enrutamiento *"unicast"*, *"carry and forward"*, *"geocast"*, híbridos, protocolos aplicados a MANETS, entre otros.

Una parte importante de las propuestas de protocolos de red son de tipo diseminación y *broadcast*, lo anterior debido a que la motivación principal por parte del gobierno y academia, es el desarrollar aplicaciones vehiculares que brinden seguridad en el camino y tráfico inteligente colaborativo. Ejemplos de protocolos de diseminación específicos para VANETs pueden encontrarse en los trabajos de Benslimane (2004), Wischhof *et al.* (2003), Delot *et al.* (2010), Eichler *et al.* (2006) y Wischhof *et al.* (2005).

Los protocolos de enrutamiento unicast son aquellos en los cuales hay intercambio de mensajes desde un nodo fuente hacia uno destino, por ejemplo CAR (Naumov y Gross (2007)), Schnaufer y Effelsberg (2008), GpsrJ+ (Lee *et al.* (2007)), CBRP (Li *et al.* (2009), entre otros; este tipo de protocolos a su vez se pueden dividir en topológicos o geográficos.

Los protocolos de enrutamiento basados en topología utilizan información sobre los enlaces que forman los nodos de red para encaminar el paquete. Por ejemplo el trabajo de Pei *et al.* (2000).

Los protocolos de enrutamiento geográficos o basados en posición Naumov y Gross

(2007); Lee *et al.* (2007), utilizan información sobre la posición de los nodos intermediarios y destino de la ruta para encaminar el paquete; estos protocolos han tenido mucha popularidad en las redes VANETs debido a la disponibilidad y popularidad de la tecnología GPS en los vehículos.

Los protocolos *carry and forward*, o también conocidos como protocolos tolerantes a retraso (DTN: *"Delay Tolerant Protocols"*), fueron diseñados para lidiar con el alto índice de desconexiones en redes donde existe baja densidad de vehículos. En estos, los nodos retienen el mensaje cuando no existe la ruta al destino, y pasan el mensaje cuando un potencial retransmisor aparece en la vecindad inalámbrica; dicho de otra manera el mensaje es transportado por la locomoción del vehículo. Como ejemplos de este tipo de protocolos están VADD (Zhao y Cao (2006)), GeOpps (Leontiadis y Mascolo (2007)), Ohta *et al.* (2011).

En los protocolos de red tipo *geocast*, su objetivo es diseminar el mensaje a todos los nodos alrededor de un área geográfica o zona de relevancia. Variantes de *geocast* involucran tener el nodo fuente fuera del área geográfica y enrutar el mensaje hacia allá antes de comenzar la diseminación, otra variante se asegura de seguir diseminando el mensaje en el área geográfica por un tiempo determinado. Los trabajas realizados por los autores Wu *et al.* (2004), Ibrahim *et al.* (2009) y Soares *et al.* (2012) son ejemplos de protocolos *geocast* aplicados en VANETs.

Lochert *et al.* (2003), Cheng y Rajan (2003), y Tian *et al.* (2003) realizaron propuestas de protocolos de red que usan información sobre la red carretera (mapas digitales) para mejorar su desempeño y aprovechan los cruces de calles para mover el mensaje. Otros trabajos (Su *et al.* (2001); Singh *et al.* (2003)) proponen algoritmos para predecir el estado a futuro de la topología de red y poder reconstruir las rutas pro-activamente antes de que se rompan para contrarrestar el problema de cambios de los rápidos en la misma, típico en redes VANETs.

Existen propuestas de protocolos de red híbridos como solución para entornos VANETs, por ejemplo aquellos que combinan tolerancia a retraso y basados en posición, tales como los trabajos de Cheng *et al.* (2010) y TO-GO (Lee *et al.* (2010)). Además existen propuestas híbridas combinando enrutamiento basado en información de topología y de posición, tales como los trabajos de Lochert *et al.* (2003), Cheng y Rajan (2003), and Tian *et al.* (2003).

Asimismo debemos añadir aquellos protocolos de capa de red que ya fueron propuestos en la investigación de MANETs, los cuales pudieran ser modificados para elevar su desempeño aún con las características particulares de las VANET, por ejemplo las propuestas CAR (Naumov y Gross (2007)), la propuesta de protocolo de enrutamiento de Schnaufer y Effelsberg (2008), GpsrJ+ (Lee *et al.* (2007)); lo anterior adiciona un catálogo bastante amplio de propuesta de protocolos de capa de red hacia las VANETs.

La mayor parte de los protocolos de enrutamiento (unicast) en MANET son basados en topología, estos protocolos son clasificados en dos grandes grupos: reactivos y pro-activos. Los reactivos establecen las rutas cuando estas se requieren, DSR (Johnson *et al.* (2007)), TORA (Park y Corson (1997)) y AODV (Perkins *et al.* (2003)) son ejemplos de estos protocolos. Los pro-activos actualizan continuamente las rutas a todos los destinos a través del intercambio de mensajes de control periódicos, ejemplos de este tipo de protocolos tenemos a DSDV (Perkins y Bhagwat (1994)), OLSR (Clausen y Jacquet (2003)), TBRPF (Ogier *et al.* (2004)).

Ejemplos protocolos de enrutamiento para MANET basados en posición tenemos a GPSR (Karp y Kung (2000)), CBF (Fübler *et al.* (2003)), LAR (Ko y Vaidya (2000)). Propuestas de protocolos de enrutamiento *"multicast"* para MANETs, en los cuales se enruta desde un nodo fuente a múltiples destinos, tenemos a MAODV (Royer y Perkins (1999)), ODMR (Yi *et al.* (2002)) y MOLSR (Jacquet *et al.* (2001)).

Para mayor información sobre los protocolos de red en VANETs se puede consultar los trabajos del estado del arte de Chennikara-Varghese *et al.* (2006) y de Li y Wang (2007).

2.3 Retos y complicaciones en propuestas de protocolos de capa de red en VANETs

Los entornos de comunicación vehicular son caracterizados por vehículos de alta movilidad, cambios extremadamente frecuentes en la topología, y una gran variación en el número de vehículos en cierta región Schoch *et al.* (2008); Adler *et al.* (2006). Estas y otras características dinámicas en las condiciones de la VANET causan que el diseño de protocolos de capa de red se torne complicado debido a que estos protocolos deben adaptarse continuamente a estas condiciones poco fiables, no atender este aspecto ocasiona baja eficiencia y desempeño de las aplicaciones vehiculares.

A diferencia de otras redes inálambricas móviles tales como MANET o WSN, en VANETs muchas de estas características dinámicas tienen cambios más extremos y en poco tiempo, lo cual hace más critico el problema. Por ejemplo, la velocidad del nodo puede variar desde cero km/h en nodos que son unidades estacionarias a lado de la carretera, o inclusive cuando los vehículos están parados en un embotellamiento, hasta más de 130 km/h en carretera. Es por esto que el atender alguna o varias de estas características dinámicas es lo que hace que las propuestas de protocolos de red sean específicos para las VANETs.

Schoch *et al.* (2008) mencionan algunas condiciones dinámicas claves de la red VANET, tales como la densidad de la red, la velocidad del nodo, la heterogeneidad de los nodos, los patrones de movimientos de los vehículos. Viendo esto desde el punto

de vista de cómputo consciente del contexto (Abowd *et al.* (1999)), estas características dinámicas de la red se traducen a información de contexto útil para un protocolo de capa de red.

A estas propuestas de protocolos de red que cuentan con mecanismos para adaptarse en tiempo de ejecución a las características dinámicas de la red y de las aplicaciones las denominamos como protocolos adaptativos, hablaremos más de ellos en el siguiente capítulo.

Varios autores como Schoch *et al.* (2008), Ramdhany *et al.* (2009), Festag *et al.* (2008) y Nundloll *et al.* (2009) han resaltado la necesidad de tener una capa de red con varios protocolos de diferente tipo. La razón de esta necesidad se da a causa de los diferentes requerimientos de comunicación de las aplicaciones vehiculares (Schoch *et al.* (2008)), las cuales no se pueden satisfacer con un solo tipo de protocolo de red o resulta más complicado buscar un protocolo de red que puede satisfacer a todo.

Tener un sistema vehicular que se desempeñe bien en una gran cantidad de aplicaciones vehiculares es deseable, debido a que se le ofrece una mayor cantidad de servicios al conductor y por ende se vuelve más atractivo comprar un vehículo con estos sistemas o instalarlo después de comprarlo; esto a su vez ayuda al crecimiento de la red VANET ya que habrá más vehículos que participen en la comunicación.

Otra causa por la cual se promueve la utilización de múltiples protocolos de capa de red, es debido a los cambios en la condiciones de la red (Ramdhany *et al.* (2009)), las cuales pueden requerir utilizar un tipo de protocolo de red en ese caso.

Por ejemplo, aplicaciones tales como informe de accidente o auto averiado se prestan para utilizar protocolos tipos *geobroacast* o diseminación de información, ya que la información debe ser transferida lo más rápido posible a un conjunto de vehículos alrededor

del nodo fuente. Por el contrario, aplicaciones como proveer servicio de Internet e informe de condiciones de camino hacia una autoridad central de tráfico, no resultan adecuados para utilizar protocolos de red tipo diseminación, debido a que el destino del mensaje es de gran importancia y la comunicación es de tipo uno-a-uno (un nodo fuente hacia uno solo nodo destino); en este caso, sería más adecuado utilizar protocolos de capa de red tipo unicast.

En referencia a seleccionar el conjunto de protocolos de capa de red adecuado, Schoch *et al.* (2008) proponen un conjunto de ellos, a los que les llaman "patrones de comunicación"; alguno de estos tipos de protocolos son diseminación, *unicast* y *carry and forward*. Los autores consideran que estos patrones de comunicación pueden satisfacer la necesidad de todas las aplicaciones vehiculares ya que resultan de un análisis del estado del arte de las aplicaciones vehiculares.

Otros autores (Nundloll *et al.* (2009); Festag *et al.* (2008)) también han propuesto diferentes tipos o categorías de protocolos de red. Ramdhany *et al.* (2009) inclusive proponen que el conjunto de protocolos de red que se utilicen se ejecuten de manera simultánea en tiempo de ejecución.

Desde el punto de vista de integración entre los sistemas de infoentretenimento de los fabricantes y la investigación y los trabajos de investigación científica o proyecto de gobierno, se necesitan por lo menos un protocolo de red tipo *unicast* para las aplicaciones de infoentreteniemiento, que necesitan comunicaciones V2I, y otro protocolo tipo diseminación para aplicaciones de seguridad en el camino, que necesitan comunicaciones V2V.

Por último, tener un conjunto de protocolos de red en el sistema vehicular permite explotar situaciones en donde una aplicación pueda seleccionar qué protocolo de red utilizar dependiendo de las condiciones de contexto, o cambiar a utilizar otro protocolo

de red cuando el actual tenga un mal desempeño en las condiciones de contexto del momento.

2.4 Resumen del capítulo

En este capítulo se describieron las características de las redes VANETs que permite a los vehículos habilitados comunicarse entre sí. Se presentaron los modos de comunicación V2V y V2I para aplicaciones vehiculares, las clasificaciones que proponen diferentes autores para éstas y su importancia como la seguridad en el camino.

Se resaltó que los esfuerzos de investigación en el área de VANETs se centra en los aspectos de capa de red, por lo que se describieron los tipos de protocolos de capa de red que existen actualmente, entre éstos los de diseminación y *broadcast*, enrutamiento *unicast*, *carry and forward*, entre otros.

Se habló sobre los retos y complicaciones para diseñar protocolos de capa de red que se adapten a la alta movilidad de las redes VANETs, los cambios frecuentes en su topología y demás condiciones poco fiables que se presentan en estas redes.

En el siguiente capítulo se presentarán las propuestas de protocolos de red adaptativos que existen actualmente y los problemas que presentan. Además las arquitecturas genéricas de adaptación y aquellas que proveen contexto a las aplicaciones móviles.

Capítulo 3

Antecedentes y trabajo previo

Este capítulo describe los trabajos realizados sobre protocolos de red que implementan mecanismos adaptativos en redes VANETs y MANETs. Adicionalmente se presentarán las propuestas de arquitecturas para construir sistemas, *frameworks* y *middlewares* orientados a proporcionar propiedades de adaptación.

3.1 Propuestas de protocolos de red adaptativos

Los protocolos de red adaptativos contienen mecanismos o algoritmos que le permiten adaptarse a las distintas características dinámicas de la red o de las aplicaciones en tiempo de ejecución. Debido a los diferentes aspectos dinámicos de las redes inalámbricas móviles (MANETs y VANETs), existen diversas propuestas de protocolos adaptativos para estas redes, las cuales han probado su efectividad para lidiar con entornos cambiantes.

Como ejemplos de protocolos de red adaptativos para VANET tenemos a: Adler *et al.* (2006) en donde adaptan el área y tiempo del protocolo de diseminación, Zhao y Cao (2006) cambian el algoritmo de enrutamiento y calibran los parámetros del enrutamiento, y Eichler *et al.* (2006) adaptan la prioridad para mandar el mensaje con base en el beneficio potencial del mismo. La Tabla 1 muestra otras propuestas de protocolos de red adaptativos para VANETs y MANETs, y describe como serían implementadas con base en el modelo del adaptación propuesto en este trabajo (véase la sección 4.1).

Por otro lado, existen otras áreas de las ciencias de la computación que también tratan aspectos dinámicos mediante propuestas de mecanismos adaptativos. Por ejem-

Tabla 1. Análisis comparativo de algunas propuestas de protocolos adaptativos de la literatura

Protocolos adaptativos	Campo de investigación	Elementos de contexto utilizados	Acciones de adaptación utilizados	Algoritmo de adaptación utilizado	Modos de adaptación utilizados
Adler *et al.* (2006)	Redes VANET	Sin elementos de contexto en específico, enfoque general por parámetros de relevancia	Decisión de forward, prioridad para mandar mensaje, acción de adaptación especial filtro	Función de relevancia de mensaje definida por los autores	Send, Receive y Forward
Delot *et al.* (2010)	Redes VANET	Vector de movilidad y dirección del evento y del vehículo	Decisión de forward, utilización de acción de adaptación especial filtro	Función de probabilidad de encuentro definida por los autores	Forward
SHARP (Ramasubramanian *et al.* (2003))	Redes MANET	El overhead del paquete, la tasa de pérdida de paquetes y el jitter entre nodos vecinos	Radio de la zona de cada nodo destino	Tres heurísticas para calcular cada métrica percibida, después modificar el radio de la zona cuando cierto umbral es alcanzado	CUPIB
AZRP (Giannoulis *et al.* (2004))	Redes Ad Hoc	Fracaso en enrutamiento y el número de nodos en la zona	Radio de zona e intervalo de actualización de rutas	Decisión de modificar el radio de la zona si resultado está afuera del umbral	CUPIB y Send
Zhao y Cao (2006)	Redes Ad Hoc	Sin elementos de contexto en específico, solo da ejemplos tal como el grado de ocupado del canal radio	Seleccionar módulo de enrutamiento, calibrar parámetros del algoritmo de enrutamiento y ajustar métricas de enrutamiento	No tiene algoritmos específicos, da ejemplos de algoritmos con mecanismos con un umbral	CUPIB
TAPR (Abuelela y Olariu (2007))	Redes VANET	Condiciones de tráfico local	Cambio entre protocolos enrutamiento carry and forward y unicast	Detectando la presencia de una ruta al siguiente cluster en el camino	Forward y Send
Eichler *et al.* (2006)	Redes VANET	Sin elementos de contexto en específico, enfoque general basado en parámetros de contexto	Prioridad para enviar los mensajes	Función de beneficio del mensaje definida por los autores	Send
Al-Doori *et al.* (2010)	Redes VANET	Dirección del camino y velocidad del vehículo	Frecuencia de mensajes broadcast de HELLO	Umbrales entre valores de velocidad y algoritmo de detección de cambio de dirección	CUPIB
Mariyasagayam *et al.* (2009)	Redes VANET	Densidad de nodos vecinos	Decisión de forward, utilización de acción de adaptación especial filtro	Algoritmo "Forwarding sectors" definido por los autores	Forward
Chen *et al.* (2012)	Redes VANET	Tiempo de semáforos, máxima velocidad de carreterra, velocidad de nodos vecinos y el "predicting slope"	Decisión de forward, use of utilización de acción de adaptación especial switcher	Algoritmo basado en el valor de la "predicting slope" y tres escenarios de luces de semáforos	Forward

plo en temas como adaptación de calidad de vídeo de acuerdo a los cambios en la conexión a la red (Rejaie *et al.* (1999)), adaptación de una interfaz gráfica (GUI) basado en la resolución de la pantalla del dispositivo (Safi *et al.* (2011)), la reconfiguración de los componentes que forman un sistema en tiempo de ejecución (Ramdhany *et al.* (2009); Popovici *et al.* (2011)), etc. Otras se centran en la idea de adaptación, por ejemplo la reflexión computacional ("computational reflection"), programación orientada a aspectos dinámicos, middleware basado en políticas ("policy middlewares") y cómputo autonómico ("autonomic computing"). Estas propuestas e ideas sobre adaptación pueden ser explotadas y enfocadas hacia la adaptación de un protocolo de capa de red.

3.2 Problemática de las propuestas de protocolos adaptativos actuales

Las diferentes propuestas de protocolos adaptativos en redes VANETs y MANETs presentan distintas problemáticas las cuales se describen a continuación. En primer lugar, se encontró que la mayor parte de las propuestas están autocontenidas, es decir, que las propuestas son descritas como otro nuevo protocolo de capa de red que maneja de mejor manera el dinamismo de la red, pero no se enfocan en relacionar sus soluciones o mecanismos de adaptación con los de otras propuestas de protocolos adaptativos. De igual manera estos trabajos previos tampoco proponen un modelo de adaptación en común entre todas las propuestas. Lo anterior dificulta promover la reutilización de partes de la solución o ideas de la adaptación.

En segundo lugar, al explorar el estado del arte en protocolos de capa de red en VANETs, notamos que el aspecto dinámico de estas redes es demasiado amplio y la mayoría de las propuestas solo atacan una parte de esta dinamicidad. Estas propuestas se enfocan en adaptar con base en contextos o información de la red específicos, optimizados para escenarios particulares (carretera, ciudad, baja densidad de nodos en la red, etc.), y dirigidos hacia un protocolo de red definido. Adicionalmente, todas las propuestas solucionan el problema de la dinamicidad de la red utilizando una clase de proceso de adaptación, el cual plantea conceptos diferentes y terminología para describir su solución (por ejemplo protocolos híbridos, adaptativo, consciente del contexto, etc). Lo anterior también dificulta hacer comparaciones entre las propuestas y reutilizar o generalizar ideas de la adaptación propuesta.

Por las razones anteriores, este trabajo de tesis se centra en promover la generalización de las soluciones o mecanismos de adaptación dentro de los protocolos adaptativos, al tratarlos como un módulo independiente de la funcionalidad del protocolo de capa de red y de las variables de contexto que utilicen como información de entrada.

Para lograr lo anterior este trabajo propone un sistema en forma de un *middleware*, que brinde mecanismos de adaptación a la capa de red de un sistema vehicular.

En la siguiente sección hablaremos de las propuestas de sistemas que manejan mecanismos de adaptación que existen en la literatura.

3.3 Arquitecturas genéricas de adaptación

En esta sección se presentarán los trabajos del estado de arte que proponen un sistema o *middleware*, el cual tiene como objetivo mejorar el desempeño de los protocolos de comunicación y aplicaciones móviles en cuanto a ciertas condiciones dinámicas.

3.3.1 PROSE + MIDAS

En Popovici *et al.* (2003) se presenta una plataforma para que los dispositivos móviles extiendan o modifiquen dinámicamente la funcionalidad de la aplicación. La idea es dejar al entorno adaptar pro-activamente la aplicación en vez de forzar a la aplicación a adaptarse a cada posible entorno. A través de esta plataforma los dispositivos móviles adquieren en tiempo de ejecución las extensiones de funcionalidad que se necesitan para trabajar correctamente en el entorno actual. Adicionalmente, los usuarios de la aplicación en el dispositivo móvil pueden seleccionar explícitamente la extensión disponible para el entorno actual.

Las extensiones son locales en tiempo y en espacio, son activadas solo en un sitio específico y por el tiempo que son necesarias. Es decir, la información de contexto en las que se basan las adaptaciones son el tiempo y la localización.

Al mecanismo para permitir a las aplicaciones ser extendidas se le llamó PROSE, este mecanismo es genérico en cuanto a que no depende de una aplicación en particular y puede darse en cualquier punto de la aplicación. PROSE se basa en el concepto de programación orientada a aspectos (AOP) y aprovecha el hecho de que las máquinas virtuales de Java tienen un compilador *"just-in-time"*. PROSE trabaja con una máquina virtual de Java modificada (una en cada nodo) para permitir la inserción de extensiones (aspectos en AOP) en tiempo de ejecución, las extensiones son manejadas como entidades Java de primera clase.

La administración de extensiones es provista por la arquitectura MIDAS, esta arquitectura permite que las extensiones provengan de una estación base (de manera centralizada) o que provengan de nodos vecinos (modo auto configurativo). Evitando así que los dispositivos sean forzados a cargar todo lo que necesitan durante su tiempo de vida operacional.

MIDAS también provee una capa de seguridad para asegurar que la extensión se origine de una fuente confiable y que no acceda a recursos no autorizados.

3.3.2 CARISMA

CARISMA (Capra *et al.* (2003)) es un *middleware* de cómputo móvil que aprovecha el principio de reflexión para mejorar la construcción de las aplicaciones móviles los cuales necesitan adaptarse a cambios en el contexto o entorno, tales como variaciones en el ancho de banda de la red, memoria y nivel de batería.

La parte central de este enfoque es hacer explícitos ciertos aspectos de la representación interna del *middleware*, y por ende accesible a las aplicaciones. Las aplicaciones pueden inspeccionar dinámicamente el comportamiento del *middleware* y además

adaptarlo dinámicamente por medio de meta-interfaces, las cuales permiten la modificación de la representación interna que previamente se hizo explícita.

Las aplicaciones especifican qué recursos les interesa y qué comportamientos adoptar en contextos particulares. El *middleware* entonces mantiene en representación de las aplicaciones, la actualización de la información de contexto, detecta cambios de interés para estas y reacciona de acuerdo a la situación.

El modelo que adopta CARISMA, supone que el comportamiento del *middleware* con respecto a un servicio particular es determinado en cualquier momento por una política. Estas políticas se definen por medio de perfiles de la aplicación, las cuales pueden ser cambiadas de manera dinámica a través de una interface de programación de aplicaciones (API, por sus siglas en inglés) reflectivo.

Los perfiles son pasados al *middleware*; cada que un servicio es invocado, el *middleware* consulta el perfil de la aplicación que lo solicita, consulta el estatus de los recursos de interés de la aplicación misma que vienen especificados en el perfil, y determina cuál política puede ser aplicada en el contexto actual, por ende releva a la aplicación de hacer estas acciones.

Puede existir un conflicto cuando diferentes políticas son utilizadas en el mismo contexto para entregar un servicio. Cuando ocurre un conflicto, se ejecuta un mecanismo de resolución basado en técnicas micro-económicas; en el cual el *middleware* juega el papel de un subastador y las aplicaciones son agentes compitiendo por bienes (que se ejecute su política). El objetivo del *middleware* es seleccionar la política que satisfaga la mayor cantidad de aplicaciones involucradas en un conflicto. De esta manera, el *middleware* se enfoca en proveer un protocolo que habilite a los dispositivos móviles a negociar por servicios disponibles en el entorno al adaptar las políticas de acceso a estos servicios. CARISMA enfoca su contribución hacia describir a detalle el mecanismo de

resolución de conflictos.

3.3.3 MANETKit y propuesta de Nundloll et al.

En MANETKit (Ramdhany *et al.* (2009)) se propuso un *framework* de tiempo de ejecución orientado a componentes para la implementación, emplazamiento y reconfiguración dinámica de protocolos en enrutamiento ad-hoc. MANETKit permite el despliegue de múltiples protocolos de manera simultánea, y elegir entre ellos de acuerdo a las condiciones de operación actual. Además de poder cambiar entre protocolos, el *framework* permite la generación de variaciones o híbridos de los protocolos a través de una reconfiguraron dinámica más minuciosa.

MANETKit hace uso de un patrón Control-Reenvío-Estado (CFS: *"Forward-Control-State"*) y tuplas <eventos-requeridos, eventos-proporcionado> para permitir a los protocolos encajar en una pila de protocolos, ser compuestos de diferentes maneras y ser fácilmente reconfigurados dinámicamente. La comunicación entre unidades CFS dentro de una entidad MANETKit, por ejemplo el flujo de paquetes o la información de contexto es llevada a cabo a través de eventos.

El sistema MANETKit provee un conjunto de tipos de eventos relacionados a la información de contexto del sistema tales como la calidad del enlace, la calidad de la señal, la relación señal a ruido, el ancho de banda disponible, la utilización de CPU, consumo de memoria y niveles de batería. Además, los protocolos pueden también proporcionar eventos de contexto específicos.

Dentro de una reconfiguración dinámica, MANETKit define el monitoreo del contexto y la promulgación de una reconfiguración. Sin embargo, no trata o define el paso de la toma de decisión (obtener la información de contexto y pasarla por reglas

de acción evento-condición), esta toma de decisiones lo deja a responsabilidad de las aplicaciones o software de más alto nivel.

La propuesta de Nundloll *et al.* (2009) utiliza el mismo *framework* propuesto por MANETKit pero enfocado a redes VANETs. Al igual que MANETKit, este *framework* encapsula un protocolo de capa de red en un componente, éste a su vez es dividido en sub-componentes utilizando el patrón Control-Reenviar-Estado (CFS). La propuesta de Nundloll *et al.* (2009) adapta y reconfigura los protocolos haciendo cambios a dos niveles: agregar/quitar componentes (cambiar entre protocolos de red), y cambiar sub-componentes dentro del protocolo.

Adicionalmente a lo propuesto en MANETKit, la propuesta de Nundloll *et al.* (2009) identifica operaciones comunes de los protocolos de capa de red en donde se pueden realizar adaptaciones, estas operaciones fueron: *"send"*, *"receive"*, en la calendarización de tareas y en el proceso de descubrimiento de vecinos. El *framework* propuesto en MANETKit también ha sido utilizado para redes inalámbricas de sensores (Grace *et al.* (2006)).

3.3.4 CATS

CATS (Popovici *et al.* (2011)) propone un *framework* para dispositivos móviles relacionados al transporte, el cual permite construir aplicaciones flexibles a la información de contexto del dispositivo. CATS construye una aplicación flexible constituida por un conjunto de servicios, que especifican para que situación de contexto funcionan. La adaptación ocurre al insertar, cambiar o remover servicios de tal manera que cumplan con las condiciones de contexto.

El *framework* CATS es un sistema distribuido, de tal manera que existe una instancia CATS en cada dispositivo móvil. CATS consta de 4 componentes: un manejador de

contextos, un manejador de ejecución y uno denominado como *"trader"*. El manejador de contexto se encarga de tener actualizada la información de contexto, consultando a las fuentes de contexto mediante un mecanismo publicación y suscripción.

El manejador de ejecución se encarga de administrar los servicios, es decir, es el responsable de adaptar la aplicación cambiando los servicios que la componen. Para los servicios que no se adecuen a las condiciones de contexto actual, separa el servicio y se inicia un servicio equivalente que sí cumpla con las condiciones de contexto. En caso dado que no se encuentre localmente un servicio equivalente, este se buscará en los dispositivos vecinos utilizando el componente *trader*.

El *trader* se encarga de descargar e instalar servicios que no estén en el dispositivo móvil de una fuente confiable, su objetivo es encontrar en los dispositivos vecinos un servicio apto para el contexto actual en el menor tiempo posible.

CATS propone un conjunto de contexto específico para aplicaciones móviles relacionadas al transporte, que clasifica en los siguientes grupos: contextos de dispositivo, contextos de ejecución, contextos del entorno y contextos de usuario.

Para implementar la modularización de servicios y aplicaciones, además de manejar la asociación dinámica de servicios y aplicaciones, se utilizó el *framework* OSGi. En OSGi los servicios y aplicaciones de CATS son módulos OSGi (*"bundles"*).

3.3.5 MARCHES

La propuesta MARCHES (Liu y Cheng (2008)) es un *middleware* de adaptación basado en componentes que construye aplicaciones colaborativas. Las aplicaciones consisten en un conjunto de componentes independientes, los cuales forman una cadena de com-

ponentes.

Para construir aplicaciones adaptativas en MARCHES, los desarrolladores deben proveer un archivo *"script"*, el cual está dividido en una parte declarativa y una parte que contiene la reglas de adaptación. La parte declarativa indica todos los componentes utilizados en la aplicación local y el agente del middleware. Basados en esa declaración, el agente MARCHES carga e instancia los componentes con sus parámetros de inicialización. La otra parte contiene las reglas de adaptación y cada una puede ser separada en tres secciones: un sensor, un actuador pro-activo, y un actuador reactivo el cual es opcional.

MARCHES se localiza entre la capa de red y capas *"hardware"*, y entre la capa alta de aplicación. De esta manera puede observar el entorno y dar soporte a la adaptación de aplicaciones. MARCHES es *middleware "peer-to-peer"*, ya que existe un agente del *middleware* por aplicación en cada computadora.

Como entrada para las adaptaciones de las aplicaciones, MARCHES obtiene el contexto actual mediante sensores. Cada sensor en MARCHES está construído en base a un modelo jerárquico de notificación de eventos, en donde un sensor puede ser integrado por varios eventos contextuales integrados como un árbol binario.

MARCHES también propone un protocolo de sincronización utilizando mensajes activos para que la reconfiguración dinámica (la adaptación) puede crear comportamientos consistentes a través de las cadenas de componentes distribuidas. Utilizando los mensajes activos, MARCHES soluciona el problema crítico de tener un tiempo de reconfiguración alto para realizar una adaptación de la aplicación colaborativa.

3.4 Arquitecturas que proveen contexto a aplicaciones en ambientes móviles

En esta sección presentaremos los trabajos del estado de arte que proponen un sistema o plataformas para ofrecer información de contexto a las aplicaciones u otras entidades, de tal manera que éstas utilicen dicha información para adaptar su comportamiento. Estos trabajos relegan la parte final de la adaptación a las aplicaciones, solo proponen una plataforma o sistema para proveer la información del contexto del entorno del problema.

3.4.1 RCSM

Reconfigurable context-sensitive middleware for pervasive computing (Yau *et al.* (2002)) es un middleware que provee desarrollo y soporte en tiempo de ejecución para permitir a las aplicaciones aprovechar la información de contexto para comunicarse con otros dispositivos. RCSM fue diseñado para facilitar el desarrollo de aplicaciones que requieren comunicación *"ad-hoc"* consciente del contexto.

Utilizando RCSM, una aplicación consciente del contexto es modelada como objetos sensible al contexto, los cuales consisten en dos partes: una interfaz que encapsula la sensibilidad de contexto de la aplicación (lo cual incluye la lista de elementos de contexto relevantes, acciones y mapeo entre ambos), y la implementación de las acciones que el *"software"* de aplicación debe proveer. De esta manera, el desarrollador se enfoca en implementar estas acciones, en cualquier lenguaje de programación, sin preocuparse de la supervisión, detección y análisis del contexto.

Para dar soporte a comunicación espontánea entre objetos, RCSM utiliza un modelo de comunicación descentralizada y un enfoque orientado a mensajes (CORBA, orientado a ORB), el cual permite al *middleware* descubrir nuevos objetos y establecer nuevos

enlaces de comunicación entre ellos.

3.4.2 Propuesta de Kim et al.

Kim *et al.* (2004) presentan un *middleware* consciente del contexto, este *middleware* utiliza un esquema de contexto estandarizado llamado *"Human Interaction Markup Language"* (HIML) para manipular información de contexto. El uso de HIML permite tener un lenguaje común para expresar el contexto en cualquier tipo de dispositivo, y para facilitar la interacción del usuario con el sistema de contexto al modificar la información de contexto actual.

Este *middleware* permite al usuario dar explícitamente o ajustar los parámetros contextuales desde la terminal de su dispositivo, por lo que nueva información contextual es generada desde la terminal, creada en formato HIML y guardada en una base de datos de contextos. La información de contexto acumulada es fusionada y analizada por medio de minería de datos para formar nuevos contextos que pueden ser utilizados en los dispositivos con una terminal.

3.4.3 CAMUS

El *middleware* CAMUS (Ngo *et al.* (2004)) se enfoca en proveer composición de información de contexto y propone una separación de preocupaciones entre las diferentes técnicas de sensado y de procesamiento de información de contexto. Los datos son extraídos de sensores, llamados características (*"features"*), los cuales son representados en un espacio de tuplas de características, el cual sirve como mecanismo de comunicación y de almacenamiento del *middleware*.

Las tuplas son asociadas de tal manera que permitan convertir una característica dada en un contexto, el cual es almacenado en un repositorio ontológico. La ontología contiene los conceptos de dominio y las propiedades especificadas en semántica formal,

esto habilita la categorización de las entidades de contexto en agentes, dispositivos, entornos, localización y tiempo.

CAMUS permite que se le incorporen varios mecanismos de razonamiento a manera de servicios que se conectan al *middleware*, estos mecanismos de razonamiento se utilizan para componer información de contexto proveniente de varios sensores. Por ejemplo, se le pueden conectar mecanismos de razonamiento tales como lógica difusa y redes bayesianas.

3.4.4 Semantic Space

Semantic space (Wang *et al.* (2004)) es una infraestructura de computo pervasivo que aprovecha las tecnologías de *"Web"* semánticas para dar soporte a la representación explícita, consulta y razonamiento flexible del contexto en ambientes inteligentes.

Semantic Space propone utilizar una ontología de la información de contexto que caracteriza el entorno. Esta información de contexto se identifica en tres clases de objetos: contexto de usuario, contexto de localización y contexto de entidad computacional. Además se identifica una clase de objeto conceptual para representar una actividad, la cual es un contexto de alto nivel que puede ser inferido por otros contextos.

La infraestructura de una *Web* semántica provee mecanismos para permitir a las aplicaciones obtener la información de contexto utilizando consultas tipo declarativas, inferir contextos de alto nivel utilizando reglas heurísticas, habilitar a los dispositivos a unirse dinámicamente al entorno, y observar dispositivos para abstraer de ellos la información de contexto.

3.4.5 Context widget

La propuesta Context Toolkit (Salber *et al.* (1999)) propone una representación de la información de contexto mediante el uso de *"widgets"* de contexto, y un mecanismo para proveer esta información a las aplicaciones. Esta propuesta se construye en base al concepto de un *widget* que viene de las plataformas de interfaces gráficas (GUI); de igual manera que un *widget* GUI media entre la aplicación y el usuario, un *widget* de contexto media entre la aplicación y el *software* representando al entorno.

Los *widgets* de contexto tienen un estado y comportamiento. El estado del *widget* es un conjunto de atributos que pueden ser consultados por las aplicaciones. Un *widget* de contexto está constituido de un número de componentes que pueden ser de tres tipos: generadores, interpretes y servidores. Un intérprete abstrae la información de contexto de más bajo nivel hacia información de contexto de más alto nivel, este componente permite la composición o fusión de información de contexto.

En la propuesta Context Toolkit todos los *widgets* de contexto almacenan los datos históricos en una base de datos para poder ser consultados por las aplicaciones.

La información de contexto puede ser entregada a las aplicaciones ya sea por un mecanismo de publicación y suscripción (*"publish and susbcribe"* en inglés), o mediante un mecanismo de encuesta (*"polling"* en inglés). Un *widget* de contexto permite además que se especifiquen condiciones a cumplirse para que la información de contexto sea notificada a la aplicación cuando ocurra un cambio en su valor.

Finalmente, para permitir que la mayor cantidad de aplicaciones heterogéneas puedan utilizar el Context Toolkit, este supone que trabajan arriba de TCP-IP, además de utilizar como modelo de comunicación al protocolo HTTP y XML.

3.4.6 CoSphere

CoSphere (Peddemors *et al.* (2005)) propone una arquitectura para expresar, proveer y modificar la información de contexto relacionada a la comunicación de una aplicación móvil. Esta arquitectura permite a las aplicaciones obtener la información de contexto de manera procesada y estructurada. Además, esta arquitectura contempla la existencia de múltiples tecnologías de red (Zigbee, 802.11, Bluetooth) en el dispositivo móvil, obtener información de contexto sobre ellas y poder seleccionar cuál de ellas utilizar.

El componente central de la arquitectura CoSphere es el proveedor de contexto de comunicación, este recolecta los datos de las fuentes de contexto y se encarga de agregar e inferir esta información, a un nivel de abstracción y detalle que empate con las necesidades de cada tipo de aplicación en particular.

La aplicación interactúa con el proveedor de contexto mediante un API ofrecido por éste. Una vez que la aplicación se conecta al proveedor de contexto, este último crea un componente adaptador configurado para esa aplicación en específico; el cual se encarga de agregar e inferir la información de contexto que solicita la aplicación de acuerdo al nivel de abstracción y detalle especificado. Además, el proveedor de contexto puede crear un componente actuador mediante el cual la aplicación puede modificar cierta información de contexto.

3.5 Resumen del capítulo

En este capítulo se definieron los protocolos adaptativos como mecanismos que adaptan al protocolo a las características de las redes o de las aplicaciones en tiempo de ejecución. Se presentaron los trabajos realizados sobre protocolos de red adaptativos en redes VANETs y MANETs, y el análisis comparativo de sus propuestas con referencia al tipo de red en el que funcionan, elementos de contexto en los que se basan, acciones de adaptación, algoritmo de adaptación y modos de adaptación que implementan.

Se habló de los problemas que presentan las propuestas de protocolos adaptativos actuales, entre estos que cada una presenta su propio protocolo para manejar el dinamismo de la red, y que solo atacan parte de esta dinamicidad, en lugar de proponer un modelo de adaptación común en el que puedan participar los protocolos de las diferentes propuestas, lo que dificulta la reutilización de las soluciones de adaptación alcanzadas.

Se resaltó que este trabajo de tesis se centra en tratar a los mecanismos de adaptación de los protocolos de capa de red como módulos independientes a la funcionalidad que proporciona el mismo y a la información de contexto que utilice como entrada, para lo que se propone el desarrollo de un *middleware* que brinde los mecanismos de adaptación a los protocolos de capa de red de un sistema vehicular.

Finalmente, se introdujeron los trabajos realizados sobre arquitecturas *middlewares* y *frameworks* que proporcionan propiedades de adaptación como PROSE + MIDAS, CARISMA, MANETKit y la propuesta de Nundloll *et al.* (2009), CATS y MARCHES, y aquellas que proveen contexto a las aplicaciones en ambientes móviles como RCSM, la propuesta de Kim *et al.* (2004), CAMUS, Semantic Space, Context widget, CoSphere.

En el siguiente capítulo se presenta el modelo de adaptación propuesto en este trabajo de tesis, y la arquitectura del *middleware* de adaptación.

Capítulo 4

Arquitectura de middleware de adaptación

En este capítulo se describirá la arquitectura del *middleware* de adaptación propuesta como trabajo de tesis. El capítulo está estructurado de la siguiente manera: primero se describe el modelo de adaptación utilizado como base para la arquitectura, enseguida presenta una vista general de la arquitectura de adaptación propuesta, y finalmente los detalles de los aspectos específicos de la arquitectura

4.1 Modelo de adaptación

La arquitectura propuesta en este trabajo tiene como núcleo un modelo de adaptación, el cual describe en esta sección.

El concepto de adaptación se refiere a la habilidad de un proceso o entidad de alterar su comportamiento durante su tiempo de vida, con base en las condiciones cambiantes de su entorno.

El concepto de adaptación se integra al de un "protocolo adaptativo", en este trabajo éste otro concepto se refiere a cualquier protocolo de capa de red que se adapta en base a la información del contexto del nodo o vehículo.

Una idea clave del modelo de adaptación es el de separar la funcionalidad de un protocolo adaptativo en dos partes independientes. La primera parte es la funcionalidad específica de un protocolo de capa de red, por ejemplo la definición del formato de los mensajes/paquetes, las operaciones de envío y recepción de mensajes, el mecanismo de supresión de mensajes, la actualización de la topología, la conexión a capas inferior y superior, etc. Esta funcionalidad generalmente representa la mayor parte de un pro-

tocolo adaptativo debido a su complejidad.

La segunda parte de un protocolo adaptativo es la funcionalidad que realiza las adaptaciones específicas en el protocolo, las cuales lo hacen más robusto a cambios dinámicos del entorno, nosotros llamados a esta parte las "soluciones de adaptación". Un protocolo adaptativo puede tener varias soluciones de adaptación, cada una mejorando en ciertos aspectos dinámicos al protocolo de red.

El modelo de adaptación propuesto en este trabajo combina los conceptos descritos anteriormente con el principio de separación de preocupaciones (Win *et al.* (2003)), un principio general de ingeniería en computación que promueve la separación de los diferentes intereses o preocupaciones de un problema, para después solucionarlas separándolas en módulos que no requieren información detallada sobre los demás módulos, y por último combinarlos en un resultado final. Las dos preocupaciones que se separaron en este trabajo son la funcionalidad de las soluciones de adaptación y la funcionalidad de un protocolo de capa de red; la solución completa sería un protocolo adaptativo.

El modelo de adaptación propone además la separación del concepto de solución de adaptación en varios sub-conceptos, los cuales llamaremos elementos de adaptación. La solución se divide en tres elementos:

- *Elemento de contexto:* Simboliza una pieza de información sobre el entorno, el cual es utilizado como valor de entrada al ejecutar una solución de adaptación. Encapsula las instrucciones o código que administra y provee esta información al *middleware* o algún otro agente externo. Un elemento de contexto es similar al concepto de métrica de los trabajos de Boleng *et al.* (2002) y Yawut *et al.* (2008).
- *Algoritmo de adaptación:* Este concepto encapsula las instrucciones, lógica u operaciones que utiliza la solución de adaptación. El algoritmo de adaptación

recibe como entradas un conjunto de elementos de contexto y como resultado produce cambios en la entidad adaptable. El algoritmo de adaptación es similar al concepto de "política" utilizado por Hadzic *et al.* (1999).

- *Acción de adaptación:* Simboliza una característica, elemento o acción el cual se modifica, activa, desactiva o cambia como resultado de la solución de adaptación. Esto es similar al concepto de "mecanismo" de Hadzic *et al.* (1999).

La separación de la solución de adaptación promueve aún más la modularización de un protocolo adaptativo. Un segundo beneficio es que facilita la comparación y clasificación de trabajos previos observando los elementos de adaptación que utilizan. Finalmente, esta separación promueve una rápida calibración de una adaptación mediante el simple intercambio de elementos dentro de la solución de adaptación hasta encontrar una combinación de elementos que produzca el resultado esperado.

La relación entre elementos de una solución de adaptación es modelada como un algoritmo con entradas y salidas (1).

$$\text{Algoritmo(N Valores de elementos de contexto)} \rightarrow \text{R Valores de acciones de adaptación} \quad (1)$$

El algoritmo de adaptación toma como entrada una lista de elementos de contexto, estos valores son utilizados dentro de las instrucciones y operaciones del algoritmo. El valor actual de los elementos de contexto son recolectados previamente del entorno de contextos y utilizados para construir la lista de entrada al algoritmo. Como resultado, el algoritmo produce una lista de valores de salida las cuales corresponden a unas acciones de adaptación. Finalmente estos valores de salida son transmitidos a las entidades de las acciones de adaptación, las cuales con este valor modifican el comportamiento del proceso adaptable. El proceso de ejecución de una solución de adaptación se muestra en la figura 3.

El concepto de solución de adaptación propuesto en este trabajo es similar a la ejecución de una reconfiguración dinámica de Ramdhany *et al.* (2009); en el cual

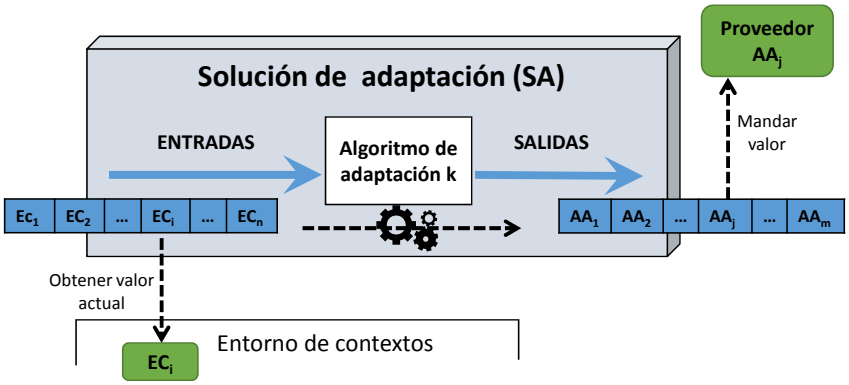

Figura 3. Descripcion gráfica de la ejecución de una solución de adaptación

una reconfiguración dinámica involucra un sistema de control cerrado con los siguientes componentes: monitoreo de contexto (elementos de contexto), toma de decisiones basado en utilizar información de contexto en reglas de evento-acción-decisión (algoritmo de adaptación), y por último la promulgación de la reconfiguración (acciones de adaptación).

Otro aspecto del modelo de adaptación propuesto, es la definición de dónde o cuándo se ejecutará una solución de adaptación durante el proceso adaptable. Se nombraron a estos lugares o puntos del proceso como "modos de adaptación". Por ejemplo, para un protocolo de capa de red, algunos lugares o puntos para ejecutar soluciones de adaptación pueden ser: cuando el nodo recibe un mensaje, cuando se modifica la tabla de enrutamiento, cada cierto tiempo, cuando se cumplen ciertas condiciones en los elementos de contexto, etc.

Para nuestro modelo de adaptación definimos solo un conjunto de modos de adaptación los cuales están enfocados a un protocolo adaptativo y creemos son suficientes para realizar las adaptaciones deseadas en este trabajo de tesis. Es posible definir un modelo de adaptación flexible, que permita definir modos de adaptación en cualquier punto del proceso (por ejemplo en Ramdhany *et al.* (2009) y en Popovici *et al.* (2003)). Sin embargo, lo anterior puede provocar cambios fuertes en el proceso, y en caso de tratarse

de un protocolo de capa de red es posible que se tenga que volver a implementar por completo. En este trabajo se optó por proponer un modelo en el cual solo se tiene un conjunto estático de modos de adaptación, esto con la intención de que una apropiada selección de este conjunto logre disminuir la tarea de portar una implementación de un protocolo de red ya existente para poder utilizarse en el middleware propuesto.

La última característica del modelo de adaptación es como se integra un protocolo de capa de red al concepto de una solución de adaptación debido a que está ligado a su acción de adaptación. Un protocolo puede ofrecer varias acciones de adaptación, cada una permite que el protocolo sea modificado por las soluciones de adaptación al enviarle un nuevo valor de la acción de adaptación. Existirán además unas cuantas acciones de adaptación especiales, las cuales no están asociados a un protocolo de red en específico, sino ofrecidas por el *middleware*; su descripción a detalle se encuentra en la sección 4.5.

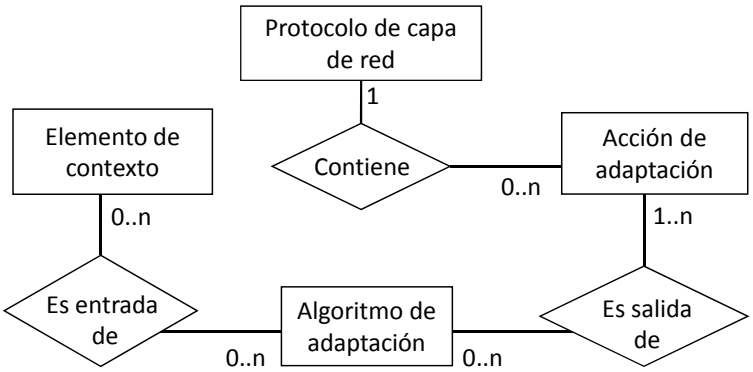

Figura 4. Diagrama entidad/relación describiendo la relación entre los elementos de una solución de adaptación

Para finalizar, en la figura 4 mostramos un diagrama Entidad/Relación que describe la relación que tienen los elementos de adaptación, y los protocolos de red.

4.2 Vista general de arquitectura de middleware de adaptación

En esta sección se describirá de manera general la arquitectura del *middleware* de adaptación propuesta, la cual fue denominada PLUGAM (*"**PLUG**-in-based **A**daptation **M**iddleware"*). Esta arquitectura está basada en el modelo de adaptación descrito en la sección anterior. Después de presentar la vista general de la arquitectura, en secciones posteriores se describe cada aspecto en detalle.

El objetivo del *middleware* PLUGAM es el proponer una arquitectura genérica; es genérica debido a que construye un conjunto de protocolos adaptativos existentes en la literatura. Las soluciones de adaptación integradas en el protocolo adaptativo, son construídas utilizando componentes externos e independientes, los cuales se llaman "componentes de extensión" en la arquitectura. Los componentes de extensión definidos en la arquitectura son: los elementos de adaptación (elementos de contexto, acciones y algoritmos de adaptación), las aplicaciones vehiculares y los protocolos de capa de red.

Después de que el *middleware* carga todos los componentes de extensión necesarios para construir una solución de adaptación, el siguiente paso es expresar la combinación de componentes que forman la solución de adaptación. Para expresarla, el *middleware* tiene que crear lo que llamaremos como "configuración de adaptación" (ver sección 4.6 para más información).

La arquitectura hace uso de la tecnología de *plug-ins* como mecanismo para encapsular y cargar los componentes de extensión al *middleware*, para esto se definieron un conjunto de tipos de *plug-ins* (véase sección 4.7).

La arquitectura PLUGAM también incluye un mecanismo para identificar de manera única un componente de extensión de todos los demás, inclusive de otros componentes del mismo tipo (por ejemplo entre dos protocolos de red o dos elementos de contexto).

El uso principal de este mecanismo de identificación, es para referenciar que componente de extensión de los que cargó el *middleware* es requerido en cada campo de una configuración de adaptación; además de otros usos los cuales se explicarán en la sección 4.4.

Para la arquitectura PLUGAM se propuso utilizar cuatro modos de adaptación: *send*, *receive*, *"forward"* y el modo CUPIB. Los modos *send*, *receive* y *forward* son en referencia al paso de mensajes entre la aplicación y el protocolo de red, durante el recorrido del mensaje del nodo fuente al destino. El modo CUPIB no está ligado a la interacción entre aplicación y protocolo de red, sino más bien es un modo que ejecuta la solución de adaptación cada cierto intervalo de tiempo. La sección 4.5 contiene una descripción más detallada de estos modos de adaptación.

La figura 5 muestra una solución de adaptación como un módulo de ejecución que contiene un algoritmo, el cual recibe como entrada una lista de valores de elementos de contexto, y produce como salida una lista de valores utilizados para ejecutar acciones de adaptación, las cuales son parametrizadas con uno de estos valores de salida.

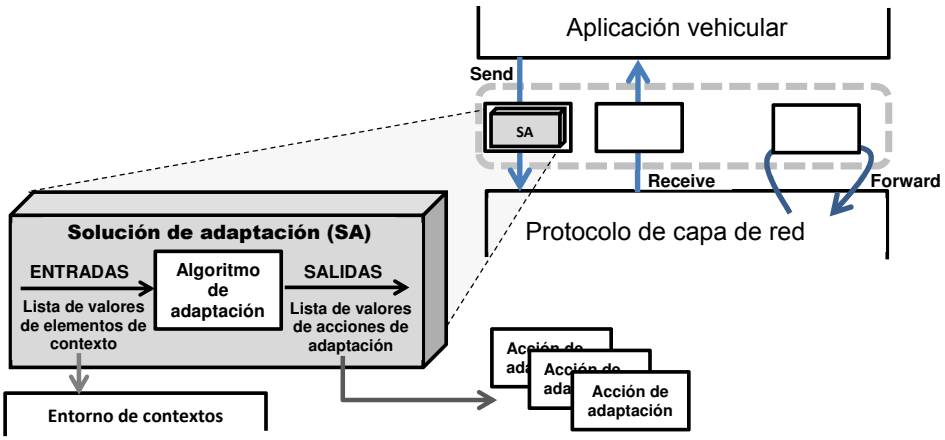

Figura 5. Representación gráfica de la arquitectura PLUGAM

Antes de ejecutar el algoritmo de adaptación, el *middleware* recopila los valores actuales de los elementos de contexto del entorno de contexto definido en la arquitectura.

Este entorno está integrado por las diferentes entidades que son fuentes de contextos, las cuales son las que ofrecen los elementos de contexto (veasé sección 4.3).

Aparte de las acciones de adaptación proveidas por los protocolos de capa de red, la arquitectura define dos acciones de adaptación de tipo especial: el conmutador (*"switcher"*) y el filtro (*"filter"*). Su descripción se presenta en la sección 4.5.

La arquitectura PLUGAM está concebida como un sistema *middleware*, situado entre la capa de red y de aplicación debido a que se necesita de la intervención de mensajes entre aplicaciones vehiculares y protocolos de capa red (véase sección 4.8).

Finalmente, en la arquitectura PLUGAM la capa de red está compuesta por múltiples protocolos de capa de red, todos ellos trabajando concurrentemente en tiempo de ejecución. Lo anterior es diferente a una pila de protocolos de comunicación común, en el cual la capa de aplicación cuenta con múltiples aplicaciones pero la capa de red solo cuenta con un protocolo de red. En este entorno de múltiples protocolos de capa de red, en tiempo de inicialización una aplicación necesita suscribirse a los protocolos que utilizará, después en tiempo de ejecución la aplicación puede enviar y recibir mensajes de los protocolos a los cuales se subscribió. Para mayor detalle véase la sección 4.9.

Esto concluye la vista general de la arquitectura PLUGAM. En las siguientes secciones se describen a detalle los diferentes aspectos de la arquitectura PLUGAM.

4.3 Representación del contexto en la arquitectura

La arquitectura PLUGAM utiliza el concepto de contexto para referirse a la información de entrada utilizada por las soluciones de adaptación. De Salber *et al.* (1999), la definición de contexto se refiere a la información que forma parte del entorno operativo del sistema y que puede ser sensado por el mismo. Por otro lado, Dey (2001)

define el concepto de contexto como cualquier información que puede ser utilizada para caracterizar la situación de una entidad, la cual puede ser una persona, lugar, objeto (o sistema de cómputo) y que es considerado relevante entre el usuario y la aplicación, incluyendo a ellos mismos.

La arquitectura PLUGAM se refiere al contexto de un nodo/vehículo o al entorno de contexto del *middleware*, como una colección de "elementos de contexto". Un elemento de contexto es proveído por una entidad dentro del nodo o *middleware* llamado fuente de contexto. Una fuente de contexto es vista como un módulo independiente, que se ejecuta de manera concurrente y proveyendo el valor más actual de los elementos de contexto que ofrece. En tiempo de ejecución el *middleware* se encarga de recuperar este valor de la fuente de contexto mediante dos mecanismos: encuesta (*polling* en inglés) y publicación y suscripción (*publish and subscribe* en inglés).

En la arquitectura se definen cuatro entidades que pueden ser fuente de contextos: un protocolo de capa de red, una aplicación vehicular y entidades que ofrecen elementos de contexto a nivel nodo-local (por ejemplo, la información de un GPS o un sensor del vehículo). Cada fuente de contexto puede proveer de múltiples elementos de contexto.

Cada elemento de contexto tiene asociada meta-información tal como nombre del elemento, el tipo de dato, etc. En tiempo de ejecución, en el tiempo t, un elemento de contexto EC tiene un valor $EC(t)$, el cual puede ser recuperado por entidades externas que tienen acceso a la fuente de contexto. En tiempo de ejecución el valor del elemento de contexto es encapsulado por un objeto ContextElementValue (ver la figura 6), el cual tiene un campo que contiene el nombre del elemento de contexto, además otro campo que contiene el tipo de dato del elemento de contexto (en caso orientado a objetos como Java este campo es una clase tipo Class).

A continuación se presentan ejemplos de los posibles valores que pueden tomar los

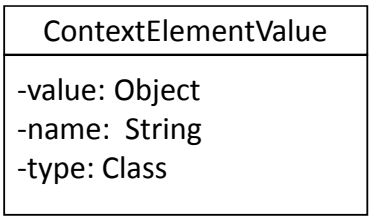

Figura 6. Diagrama de clases UML de ContextElementValue

elementos de contexto en relación a su tipo de dato:

- Conectividad 3G: Booleano (falso, verdadero).
- Tipo de camino: Cadena de caracteres (“autopista”, “rural”, “ciudad”, ...).
- Periodo de envío de mensaje de control: Entero (1, 10, 1000, ...).
- Dispersión de diseminación : Real (-1.0, 2.51, 300.7, ...)

La arquitectura PLUGAM considera que dentro del *middleware* existe un módulo llamado el manejador de contexto. Este módulo se encarga de obtener los valores más actuales de los elementos de contextos registrados en el sistema, interactuando directamente con las fuentes de contexto mediante un mecanismo de encuesta (*polling*) o publicación-suscripción. La obtención de estos valores actuales se realiza de manera concurrente a la ejecución de las soluciones de adaptación, protocolos de capa de red y aplicaciones.

La arquitectura PLUGAM permite a las fuentes de contexto cierta flexibilidad en su implementación, en el sentido que éstas pueden informar los valores más actuales ya sea sólo por el método de encuesta (es más fácil de implementar), o implementar el método publicación y suscripción de manera opcional. El manejador de contexto optará por obtener estos valores mediante el método publicación y suscripción en el caso que se implementen los dos.

Por otro lado, el manejador de contexto almacena estos valores actuales en una estructura de memoria interna de rápido acceso (por ejemplo tablas hash). La idea es

tener un entorno de contextos actualizado, de tal manera que cuando se ejecute una solución de adaptación y se necesiten los valores actuales de ciertos elementos de contexto, el manejador de contextos obtiene estos valores directamente de su estructura interna, en vez de ir recuperar estos valores dirigiéndose a cada fuente de contexto.

Se adaptó el esquema anterior con el fin de disminuir el tiempo de procesamiento al momento de ejecutar una solución de adaptación. Sin embargo, lo anterior abre la posibilidad de recuperar valores de elementos de contexto desactualizados a los que realmente se tendrían al momento de ejecutar la solución de adaptación; especialmente cuando el elemento de contexto se actualiza en el manejador de contextos con el método de encuesta (*polling*).

El costo en procesamiento del manejador de contexto puede ser reducido si las fuentes de contexto implementan el mecanismo publicación y suscripción. En el metodo de encuesta, el *middleware* necesita activar un ciclo concurrente para obtener el valor actual cada cierto tiempo. En cambio el método publicación y suscripción no necesita recursos de procesamiento adicionales por parte del *middleware*, debido a que la arquitectura ya considera a las fuentes de contexto como entidades concurrentes. Además, el favorecer al método publicación y suscripción evita tener valores no actuales al momento de ejecutar un solución de adaptación, debido a que justo cuando el valor cambia en la fuente de contexto, este inmediatamente informa al manejador de contexto del cambio.

4.4 Componentes de extensión

El modelo de adaptación propuesto en este trabajo separó el concepto de solución de adaptación en varios elementos de adaptación. Para la arquitectura PLUGAM se propuso realizar la implementación de estos elementos por separado, adicional a la de

las aplicaciones vehiculares y protocolos de capa de red. De esta manera, todas estas partes o módulos pueden ser desarrollados de manera separada e independiente entre ellas. Estos módulos independientes fueron nombrados 'componentes de extensión" de la arquitectura.

Los componentes de extensión de la arquitectura son creados independientemente del *middleware*. Por consiguiente se necesita un mecanismo para identificar de manera única cada componente de extensión y poder referirse a ellos en tiempo de inicialización y ejecución, incluso entre elementos de extensión del mismo tipo.

Fueron definidas dos maneras de identificar un componente de extensión de acuerdo con su relación con los *plug-ins*: uno-a-uno y uno-a-varios. Los componentes de extensión uno-a-uno; por ejemplo un protocolo de capa de red, aplicación o algoritmo de adaptación, son identificados de manera única tomando en cuenta la siguiente dupla (NP, HP), donde:

- NP es el nombre del *plug-in* que contiene al componente de extensión y además es el nombre del componente de extensión.
- HP es el valor *"hash"* del *plug-in* que contiene al componente de extensión.

Los componentes de extensión que tienen relación uno-a-varios entre los *plug-ins*, tales como los elementos de contexto y acciones de adaptación, estos son identificados de manera única tomando en cuenta la tupla (NP, HP, NE), donde:

- NP es el nombre de *plug-in* que contiene al componente de extensión.
- HP es el valor *hash* del *plug-in* que contiene al componente de extensión.
- NE es el nombre del componente de extensión. Se supone que este valor es único entre los otros componentes de extensión ofrecidos por el mismo *plug-in*.

Para almacenar y representar la información de identificación de los componentes de extensión, se propuso el uso de un objeto ElementIdentifier mostrado en el inciso a

de la figura 7. Este objeto es simplemente un contenedor de tres campos que contienen la información de identidad de un componente de extensión. El uso y significado de estos tres campos se muestran en el inciso b de la figura 7.

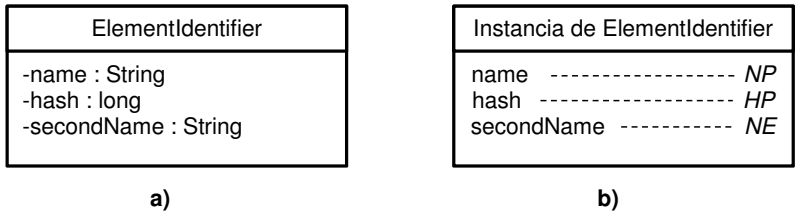

Figura 7. Definición de la clase ElementIdentifier a) y el significado de sus campos b)

4.5 Definición de las adaptaciones

4.5.1 Modos de adaptación

Dentro de la arquitectura PLUGAM se definieron cuatro modos de adaptación: *send*, *receive*, *forward* y CUPIB. Estos permiten producir adaptaciones involucrando a múltiples protocolos de capa de red, y además cambiar el valor de variables internas de los protocolos en las adaptaciones. A continuación se describe cada uno:

- *Send:* Una adaptación puede tomar lugar al momento de que se envía un mensaje. Ocurre cuando la aplicación llama al comando *send*, pero antes de que el mensaje se interne en el protocolo de capa de red.
- *Receive:* La adaptación sucede cuando un mensaje es recibido. Específicamente cuando el protocolo entrega el mensaje a la capa superior, pero antes de que la aplicación lo reciba.
- *Forward:* Una adaptación también puede tomar lugar al momento que el mensaje es retransmitido por el protocolo de capa de red, es decir, cuando éste recibe un mensaje de un nodo vecino y decide reenviarlo, justo antes que lo retransmita otros nodos.

- *CUPIB:* Finalmente se propone un modo nombrado CUPIB (**C**ontinuous **U**pdate of **P**rotocol **I**nternal **B**ehavior), el cual es utilizado para adaptaciones que modifican el comportamiento de los protocolos de capa de red, este modo se encarga de ejecutar periódicamente una solución de adaptación.

A continuación, la figura 8 muestra la ubicación de los modos de adaptación *send*, *receive* y *forward* respecto al recorrido extremo a extremo (*"end-to-end"*) de un mensaje que pasa entre las capas de aplicación y de red.

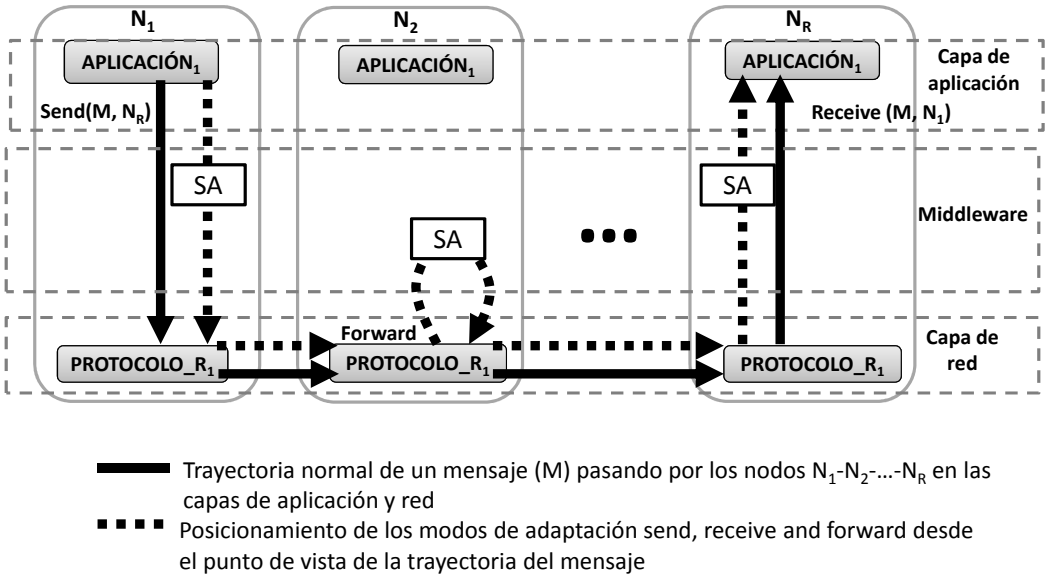

Figura 8. Ubicación de los modos de adaptación send, receive y forward desde la perspectiva del recorrido extremo-a-extremo entre las capas de red y de aplicación

Los modos de adaptación propuestos fueron deducidos luego de examinar propuestas de protocolos adaptativos de la literatura. Por ejemplo en el trabajo de Delot *et al.* (2010) los autores realizan adaptación desde el momento de recepción de mensajes a la aplicación vehicular, en los trabajos de Adler *et al.* (2006); Delot *et al.* (2010) se realizan adaptaciones en la operación *forward* de un protocolo de capa de red, en el trabajo de Eichler *et al.* (2006) los autores realizan adaptación en el momento de envío de un mensaje de la aplicación al protocolo de comunicación. Finalmente, en el trabajo de

Giannoulis *et al.* (2004) se realizan adaptaciones a un protocolo de capa de red al revaluar y actualizar el comportamiento interno del protocolo cada cierto periodo de tiempo.

El modo CUPIB es útil para situaciones de adaptación donde se requiera extender un protocolo de red mediante el ajuste de una constante interna, para hacerlo más dinámico a condiciones de contexto. Este modo permite que el *middleware* actualice de manera automática una variable de un protocolo de red con base en el contexto, esto se logra al crear una solución de adaptación en CUPIB y que asocie una acción de adaptación de salida a la variable interna de un protocolo (ver figura 9).

El periodo para volver a ejecutar una solución de adaptación en el modo CUPIB se define mediante el campo "CUPIBTime" de la configuración de adaptación. Adicionalmente, debido a que cada una de las acciones de adaptación de una solución de adaptación pueden estar asociadas a un protocolo de red (sea al mismo o diferente), es posible que una sola solución de adaptación en modo CUPIB actualice varios protocolos de capa de red al mismo tiempo.

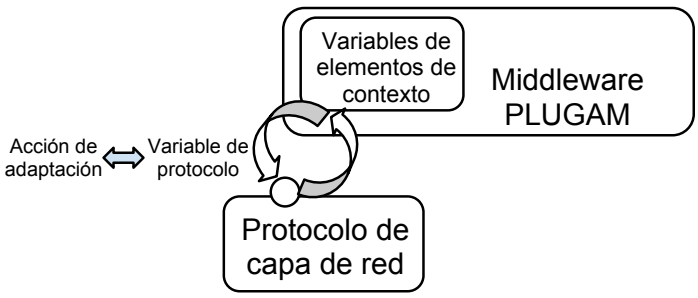

Figura 9. Modo CUPIB y la actualización de la variable de un protocolo de red

4.5.2 Acciones de adaptación especiales

Los protocolos de capa de red ofrecen diferentes acciones de adaptación, de tal manera que agentes externos (el *middleware*) puedan modificar su comportamiento. Adicional-

mente, esta definición es extendida para tomar en cuenta lo que será llamado en este trabajo como acciones de adaptación especiales. El adjetivo de especiales indica que éstas no son ofrecidas por protocolos de capa de red, sino por el mismo *middleware*.

El objetivo de estas acciones de adaptación especiales es permitir adaptaciones donde el resultado sea cambiar el comportamiento entre protocolos de capa de red y las aplicaciones. Basado en lo anterior y explorando posibles modificaciones en las interacciones entre protocolos de capa de red y aplicaciones, la definición de acciones de adaptación se extendió de esta manera:

- Acciones de adaptación de tipo protocolo
- Acciones de adaptación especiales
    - Acción de adaptación *switcher*
    - Acción de adaptación *filter*

Las acciones de adaptación tipo protocolo son ofrecidas por los protocolos de capa de red, su propósito es cambiar su comportamiento al asociar una de sus variables internas o parámetros a la acción de adaptación. Un protocolo de capa de red puede ofrecer varias acciones de adaptación, cada una asociada a una variable interna. Por ejemplo, un protocolo de capa de red tipo diseminación puede ofrecer una acción de adaptación que extienda o disminuya la propagación de la diseminación, ligándola a cambiar el valor de una variable interna del protocolo. Otro ejemplo es proponer una acción de adaptación que se asocie al valor del intervalo HELLO del protocolo de capa de red (Clausen y Jacquet (2003)).

El propósito de la acción de adaptación *switcher* es el de redireccionar un mensaje de su trayectoria predefinida cuando pasa entre la capa de red y de aplicación. El *switcher* es empleado principalmente para realizar adaptaciones que involucren cambiar el protocolo de red que se está utilizando. *Switcher* recibe como entrada un objeto ElementIdentifier, conteniendo la referencia al protocolo (o aplicación en el caso del

modo de adaptación receive) al cual se debe re-direccionar el mensaje.

La decisión de adonde re-direccionar el mensaje se toma en el algoritmo de adaptación. Después, asociando previamente al *switcher* una de las salidas del algoritmo de adaptación, este recibe la decisión final de adónde redireccionar el mensaje.

En caso que el *switcher* sea incluído en una solución de adaptación puesta en el modo de *send* o *forward*, su funcionalidad es la de re-direccionar entre los protocolos de capa de red registrados por la aplicación en el paso de inicialización del *middleware*. En caso que el *switcher* sea incluido una solución de adaptación puesta en el modo *receive*, su funcionalidad es la de re-direccionar entre las aplicaciones que hayan registrado al protocolo de red del cual viene el mensaje.

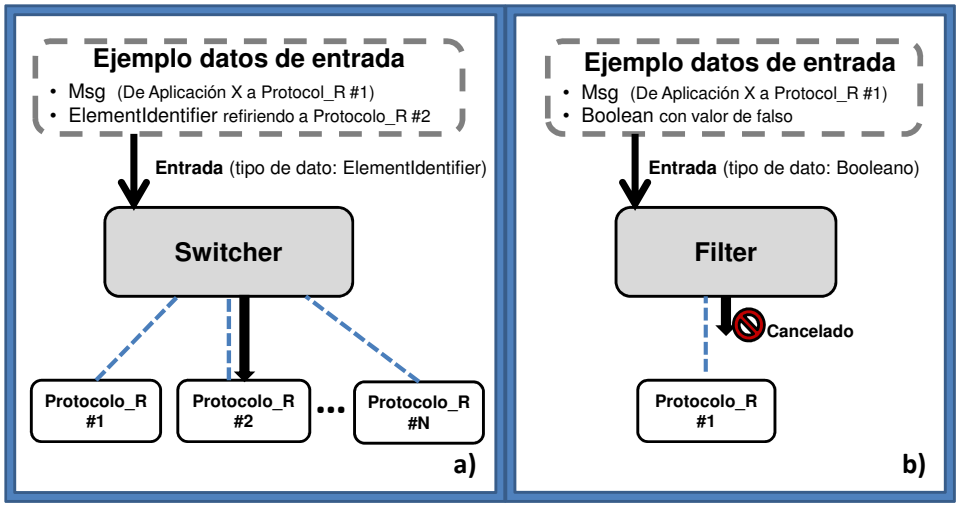

Figura 10. Representación gráfica de las acciones de adaptación switcher y filter

En el inciso a) de la figura 10 se muestra un ejemplo de como funciona la acción de adaptación *switcher*. En este ejemplo la solución de adaptación que contiene al *switcher* está puesta en el modo de adaptación *send*, la ruta predeterminada del mensaje es ir al protocolo de red #1. Sin embargo, la referencia ElementIdentifier recibida como entrada al *switcher* indica que se debe re-direccionar el mensaje al protocolo de red #2.

El propósito de la acción de adaptación *filter* es recrear un comportamiento de pasa o no pasa. Esta acción de adaptación solo puede ser utilizada en los modos de adaptación *send*, *receive* y *forward*. El *filter* recibe como entrada un tipo de dato booleano, del cual un valor de verdadero indica al *filter* que deje pasar el mensaje al protocolo o aplicación destino, y un valor de falso indica al *filter* que el mensaje debe ser cancelado y destruido.

En el inciso b de la figura 10 se muestra un ejemplo de como funciona el *filter*. En este ejemplo la solución de adaptación que contiene al *filter* recibe un valor de falso, *filter* entonces cancela y elimina el mensaje, por ende este no es transferido al protocolo de red #1.

Por último, la arquitectura utiliza objetos AdaptationActionValue para representar y transportar el valor de la acción de adaptación en tiempo de ejecución. Este objeto es muy similar al concepto de un valor del elemento de contexto descrito anteriormente; contiene los mismos campos, por lo tanto un valor de acción de adaptación puede ser de cualquier tipo de dato.

4.6 Construcción de adaptaciones en la arquitectura PLUGAM

Una vez que se cargan en el *middleware* los componentes de extensión necesarios para armar una solución de adaptación, el siguiente paso es construirla especificando los componentes de extensión que la integran. El mecanismo para describir las partes que integran una solución de adaptación es mediante la declaración de una configuración de adaptación (ver figura 11).

La mayoría de los campos de una configuración de adaptación se especifican mediante objetos ElementIdentifier. Estos objetos ElementIdentifier contienen las referen-

Configuración de Adaptación	
Campo	**Tipo de dato**
configurationName	String
adaptationAlgorithm	ElementIdentifier
contextElements:	Lista de ElementIdentifier
adaptationActions	Lista de ElementIdentifier
adaptationMode	{"Send", "Receive", "Forward", "CUPIB"}
protocol	ElementIdentifier
application	ElementIdentifier
CUPIBTime	Long integer

Figura 11. Contenido de una configuración de adaptación

cias al algoritmo de adaptación que asocia a la solución de adaptación, los elementos de contexto que se necesitan como entrada al algoritmo y las acciones de adaptación asociadas a las salidas. En una configuración también se especifica el modo de adaptación en el que se ejecutará la solución de adaptación, y se le asigna un nombre a la configuración. Adicionalmente, una configuración de adaptación contiene otros campos adicionales utilizados por ciertos modos de adaptación en particular, estos campos adicionales son los siguientes:

- *Campos protocolo y aplicación*: Estos campos son utilizados por una configuración de adaptación puesta en el modo *send*, *receive* y *forward*. Su propósito es ligar la configuración a la operación de red entre una aplicación y protocolo de red en específico; es decir, una configuración de adaptación en el modo *send* que se activa solo cuando la aplicación X llama al comando *send* del protocolo de red Y.
- *Campo CUPIBTime*: Este campo debe ser utilizado en las configuraciones de adaptación en el modo CUPIB, indica el número de milisegundos que se va a esperar para volver a ejecutar la solución de adaptación.

La arquitectura permite al usuario del *middleware* crear cualquier número de configuraciones de adaptación, lo anterior permite definir múltiples soluciones de adaptación en la capa de red. Además, las configuraciones de adaptación son estáticas, es decir,

que se definen en tiempo de inicialización y se mantienen durante todo el tiempo de ejecución del *middleware*, y por ende de las aplicaciones.

4.7 Plug-ins como mecanismo para agregar los elementos de extensión

La solución de adaptación es separada en módulos independientes llamados "componentes de extensión". La arquitectura propone el encapsulamiento de estos elementos de extensión dentro de *plug-ins*; es decir los *plug-ins* serían el mecanismo de distribución de los componentes de extensión. La idea es el desarrollar los componentes de extensión adentro de los *plug-ins*, y activarlos en el *middleware* al cargar los *plug-ins*.

Las plataformas de *plug-ins* representan un enfoque flexible para construir sistemas de software extensibles y personalizables a las necesidades de usuarios en lo individual (Wolfinger *et al.* (2006)). Estas plataformas permiten la extensión de una aplicación base con nueva funcionalidad, implementada en forma de componentes que son insertados a la aplicación base en tiempo de inicialización o en tiempo de ejecución.

Un *plug-in* es una pieza de la aplicación que una vez compilado provee un servicio específico o función en demanda. Los *plug-ins*, en general dependen de los mecanismos provistos por la aplicación base, no funcionan por sí mismos.

El uso más conocido de tecnología *plug-in* es en los navegadores *Web* de hoy en día (por ejemplo, extensiones de Google Chrome y *"add-ons"* de Intenet Explorer) y en la plataforma de desarrollo Eclipse (2004). Existen pocos casos en los que se han utilizado *plug-ins* en el desarrollo de protocolos de comunicación, por ejemplo la reconocida implementación del protocolo de red OLSR Unik-olsr (2005), la cual contiene un *framework* de *plug-in* que encapsula su funcionalidad en librerías dinámicas.

Utilizar *plug-ins* para encapsular la implementación de los componentes de extensión, facilita que los usuarios del *middleware* puedan compartir o utilizar componentes de extensión creados por terceras personas (o por un equipo diferente del proyecto); esto permite acelerar el desarrollo de nuevas soluciones de adaptación mediante la reutilización de componentes de extensión. Otro beneficio es evitar que el desarrollador de un componente de extensión interactúe con el código fuente del *middleware* cuando se desarrolla el *plug-in*, además no necesita lidiar con la compilación del *middleware*. La experiencia nos dice que aprender a extender o compilar un sistema de cómputo complejo ya desarrollado (posiblemente de miles de líneas de código separadas en varios archivos), es una tarea para nada trivial y que consume mucho tiempo, por lo que no tener que hacerlo disminuye el tiempo de desarrollo de un protocolo de red adaptativo o capa de red adaptativa.

En cuanto a las desventajas de utilizar una plataforma de *plug-ins*, una primera seria la dificultad para dar mantenimiento a la misma, en lo que respecta al manejo de versiones y compatibilidad hacia atrás con versiones existentes de *plug-ins*; específicamente cuando se requiere modificar las interfaces de los *plug-ins* o el funcionamiento de la plataforma de *plug-ins*. Una manera de eliminar esta desventaja es realizar un buen análisis y diseño inicial de la aplicación, para así minimizar que los creadores de los *plug-in* reescriban el código para que funcionen con los nuevos cambios. Otra desventaja es que habilitar un canal de comunicación entre los *plug-ins* y la aplicación principal conlleva un costo adicional de memoria y procesamiento, el cual puede ser considerable dependiendo su implementación.

La arquitectura define cuatro tipos de *plug-ins*, cada uno con su propia interfaz de servicio, la cual ofrece al *middleware* los diferentes componentes de extensión. Además, cada uno de estos tipos de *plug-in* comparten un conjunto de métodos en común, los cuales proveen meta-información sobre el *plug-in* con el propósito de identificarlo, tales

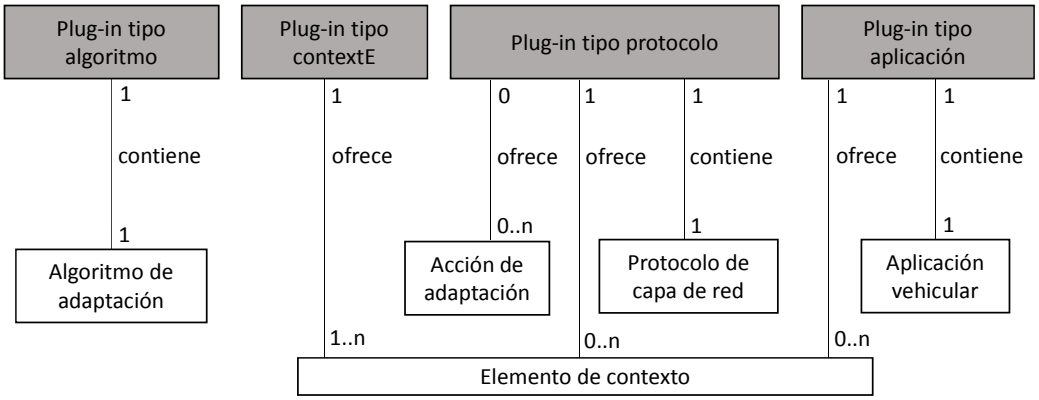

Figura 12. Diagrama UML de entidad/relación mostrando la relación entre los tipos de plug-in y los componentes de extensión

como la versión del *plug-in*, el nombre del autor, el nombre del *plug-in*, el tipo de *plug-in* que es, etc. Estos cuatro tipos de *plug-in* definidos y su relación con los componentes de extensión son mostrados en la figura 12, a continuación se describe cada uno de estos:

- *El plug-in aplicación*, el cual ofrece un componente de extensión aplicación vehicular. Este tipo de *plug-in* también puede ofrecer varios elementos de contexto, debido a que una aplicación está definida como una posible fuente contexto.
- *El plug-in contextE*, el cual ofrece múltiples componentes de extensión de elementos de contexto. El objetivo de este tipo de *plug-in* es ofrecer elementos de contexto del ámbito del vehículo o del nodo, no ligados a un protocolo de capa de red o aplicación. Por ejemplo un *plug-in* contextE que encapsule la funcionalidad de un receptor GPS, y ofrezca elementos de contexto tales como tiempo actual, velocidad del vehículo, información de localización. Otro ejemplo sería un *plug-in* contextE ofreciendo elementos de contexto relacionados a información de un sensor del vehículo.
- *El plug-in protocolo*, encapsula a un protocolo de capa de red. Además puede ofrecer múltiples elementos de contexto relacionados al protocolo, y múltiples acciones de adaptación para cambiar el comportamiento interno de éste.
- *El plug-in algoritmo*, el cual ofrece solo un componente de extensión algoritmo de

adaptación.

Para finalizar la descripción de los *plug-ins* de la arquitectura, se muestran las interfaces (API) que se definieron para cada tipo de *plug-in*; mediante las cuales el *middleware* obtiene información sobre el *plug-in*, administra su funcionamiento y obtiene los componentes de extensión. El API de los *plug-in* consiste en cuatro interfaces, una para cada tipo de *plug-in*, además de la interfaz IPlugin común para todos.

La interfaz IPlugin contiene un conjunto de métodos que proveen meta-información sobre el *plug-in*, esta interfaz debe ser implementada en todos los tipos de *plug-in*; en términos de programación orientada a objetos, las otras interfaces *plug-in* extienden de la interfaz IPlugin (ver figura 13).

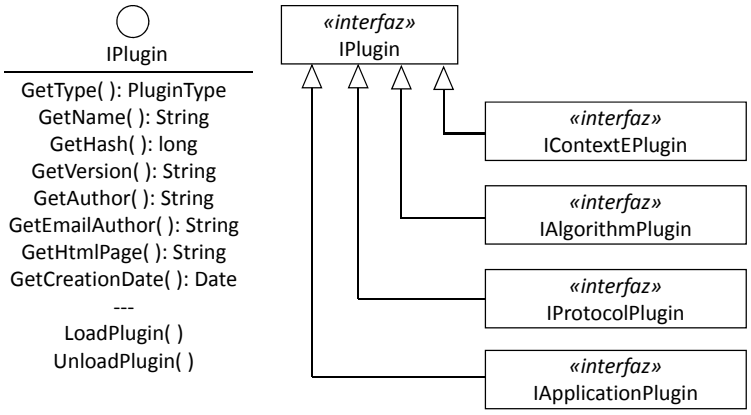

Figura 13. Definición de la interfaz y su relación con los tipos de plug-ins de la arquitectura

Por último, en la figura 14 se presenta el contenido de las interfaces de los cuatro tipo de *plug-ins* defindos en la arquitectura.

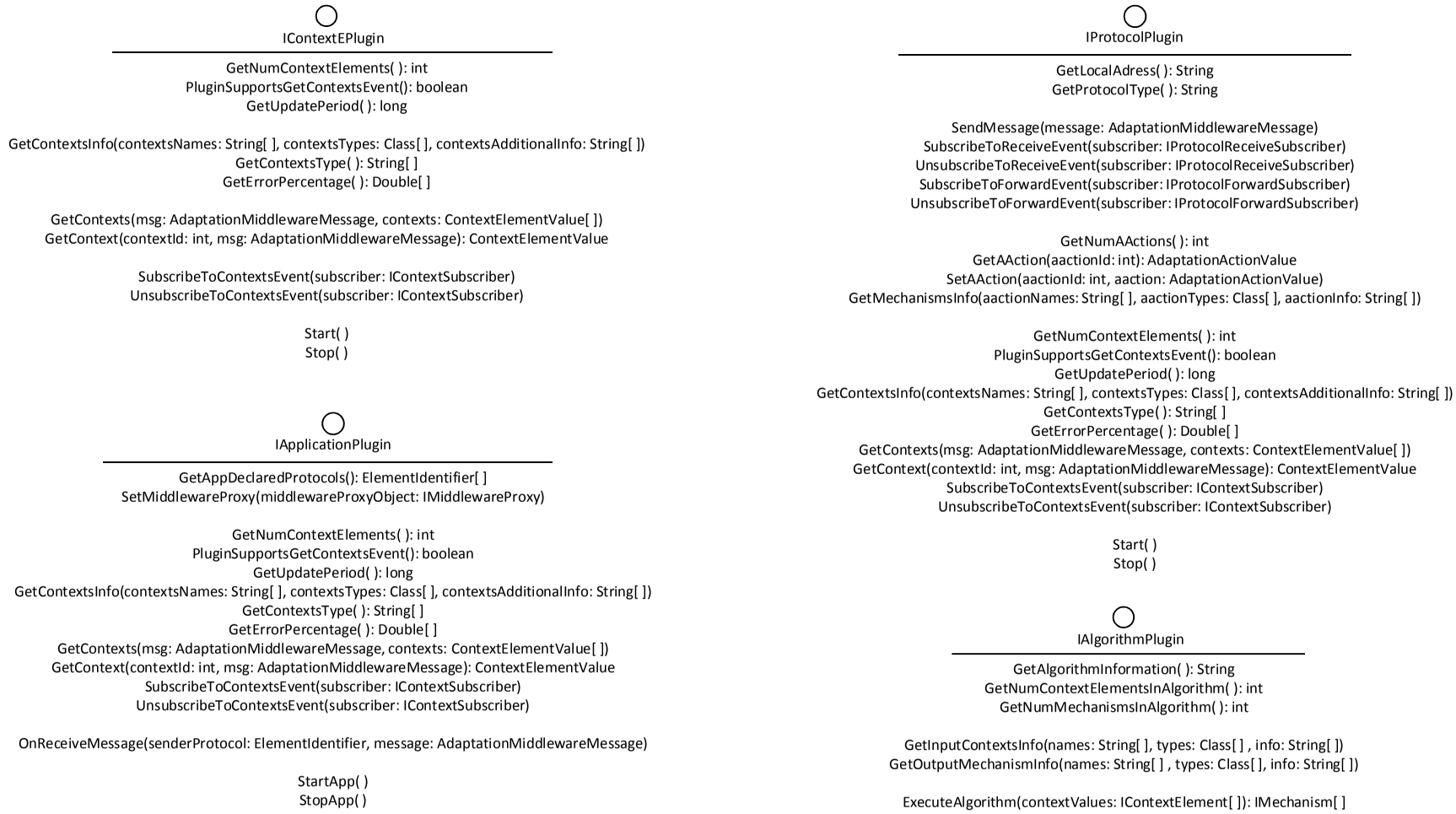

Figura 14. Diagrama de clases UML de las interfaces de los cuatro tipos de plug-in de la arquitectura PLUGAM

4.8 Aspecto middleware de la arquitectura PLUGAM

La arquitectura PLUGAM está concebida como un sistema *middleware*, situado entre la capa de red y de la aplicación, debido a que se necesita de la intervención de mensajes entre aplicaciones vehiculares y protocolos de capa red. Lo anterior con el objetivo de habilitar el funcionamiento de los modos de adaptación *send*, *receive* y *forward*. El aspecto *middleware* también sirve para ayudar a administrar el entorno de múltiples aplicaciones y protocolos de red de la arquitectura.

La arquitectura no hace referencia a las capas de transporte, sesión y presentación del modelo OSI, es responsabilidad del usuario integrar los servicios de estas capas ya sea en los protocolos de capa de red o en la aplicación. La figura 15 muestra la arquitectura del *middleware* desde el punto de vista de las capas del modelo OSI.

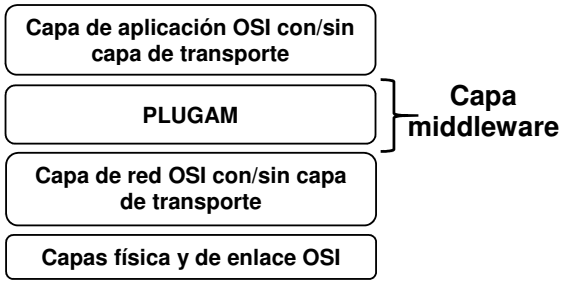

Figura 15. Posición del middleware de adaptación en el modelo OSI

Por otro lado, no existe comunicación directa entre aplicaciones y protocolos de red, sino que se comunican indirectamente utilizando los mecanismos ofrecidos por el *middleware*.

La arquitectura tiene la característica de permitir múltiples protocolos de red, lo que facilita construir un sistema vehicular con diferentes canales de comunicación inalámbrica (Wi-Fi, 802.11p, 3G, GSM, etc.). Lo anterior se logra encapsulando en un *plug-in* tipo protocolo, los protocolos de capas bajas (física y de enlace) y el protocolo

de capa de red que utilizan un mismo canal de comunicación.

En casos que existan dos protocolos de capa de red en el *middleware*, ambos utilizando el mismo canal de comunicación; por ejemplo, un protocolo de comunicación *unicast* con otro de diseminación utilizando un radio 802.11, es posible que exista interferencia por lo que el usuario del *middeware* deberá lidiar por su cuenta con éste problema.

Por último, tener una gran cantidad de protocolos de red cargados en el *middleware* no es aconsejable, debido a que estos módulos de código son muy complejos y pueden llegar a sobrecargarlo rápidamente. Encontrar el conjunto ideal de protocolos de red que sirva a todas las necesidades de las aplicaciones vehiculares, está fuera del alcance de este trabajo.

4.9 Interacción de aplicaciones en el middleware

En esta sección se describe la interacción de las aplicaciones vehiculares con el *middleware*, además se muestra como la aplicación asocia a los protocolos de capa de red que utilizará durante su ejecución.

Para comenzar es necesario aclarar que en la arquitectura PLUGAM, el *middleware* tiene el control y administra el tiempo de vida de las aplicaciones; es decir, las aplicaciones son iniciadas por el *middleware* y finalizadas cuando este termina.

Una aplicación vehicular no interactuaría directamente con los protocolos de capa de red, sino de manera indirecta utilizando al *middleware* como *"proxy"* o agente intermediario. La comunicación de dos vías entre aplicación y *middleware*, se logra mediante el uso de dos interfaces definidas en la arquitectura: las interfaces IApplicationPlugin

y IMiddlewareProxy.

El *middleware* se comunica con la aplicación utilizando los métodos de la interfaz IApplicationPlugin (ver figura 14 en la sección 4.7). En el otro sentido, la aplicación se comunica con el *middleware* mediante llamadas a métodos de un objeto *proxy*, el cual implementa la interfaz IProxyMiddleware; la definición de esta interfaz se muestra en la figura 16.

IMiddlewareProxy

Send(protocol: ElementIdentifier, mgs: AdaptationMiddlewareMessage)
GetProtocolsUseByApp(): ElementIdentifier[ ]
ApplicationFinish( )

Figura 16. Diagrama clases UML de la interfaz IMiddlewareProxy

En tiempo de inicialización, el *middleware* registra las aplicaciones disponibles, cargadas por medio de *plug-ins*. En ese momento el *middleware* crea un objeto *proxy* ligado a cada aplicación registrada, enseguida el *middleware* pasa este objeto a la respectiva aplicación vehicular mediante el método SetMiddlewareProxy de la interfaz IApplicationPlugin.

Adicionalmente, cuando el *middleware* registra a una aplicación, este primero obtiene los protocolos de red solicitados por ésta utilizando el método GetAppDeclaredProtocol de la interfaz IApplicationPlugin. Este método regresa como resultado una lista de referencias (objetos ElementIdentifier) de todos los protocolos de capa de red que necesita la aplicación. El *middleware* entonces debe hacer una búsqueda para determinar si estos protocolos de red en realidad fueron cargados en el paso anterior de inicialización. Después que la aplicación comienza su funcionamiento, esta debe utilizar el objeto *proxy* del *middleware* para interactuar de manera indirecta con los protocolos

de capa de red.

En la figura 17 se muestra un diagrama de secuencia UML que describe el proceso de registro de una aplicación por parte del *middleware*. En este proceso se obtienen los protocolos de capa de red solicitados por la aplicación, además de crear y transferir el objeto *proxy* del *middleware*.

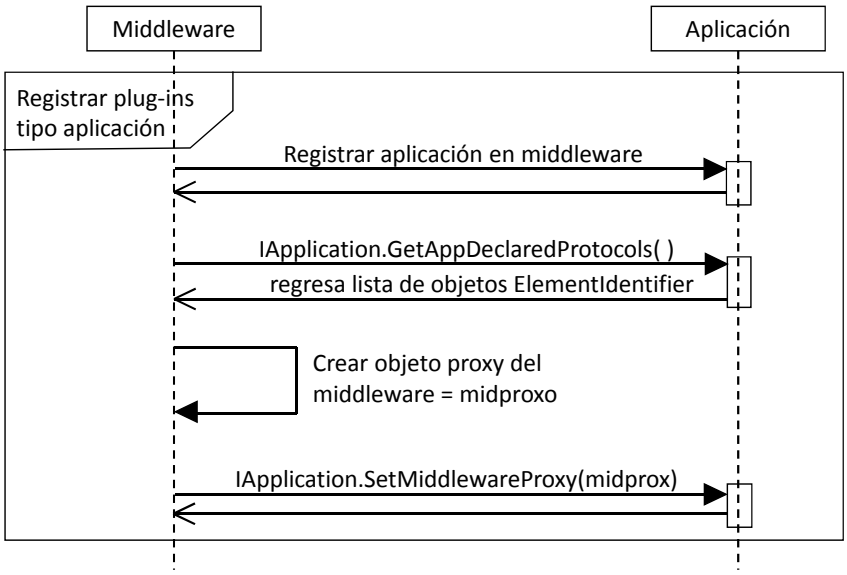

Figura 17. Diagrama de secuencia UML que describe el proceso de registro de una aplicación por parte del middleware

En tiempo de ejecución, cuando una aplicación desea enviar un mensaje utilizando un protocolo de capa de red, esta debe llamar al método *send* del objeto *proxy* del *middleware*. Este método recibe como entrada un objeto ElementIdentifier, que contiene la referencia al protocolo de capa de red seleccionado por la aplicación para enviar el mensaje. Además, este método recibe como entrada un objeto AdaptationMiddlewareMessage, el cual encapsula el contenido del mensaje (visto en la arquitectura como un simple bloque de bytes). La figura 18 muestra la definición del objeto AdaptationMiddlewareMessage.

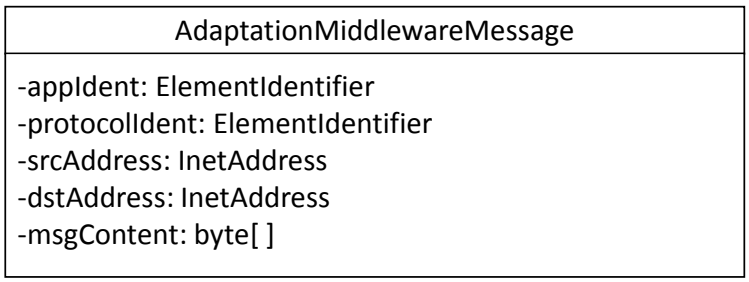

Figura 18. Diagrama clases UML de la clase AdaptationMiddlewareMessage

Por otro lado, al momento que un protocolo de red quiere transferir un mensaje hacia la aplicación correspondiente (operación *receive*), el *middleware* recibe este mensaje y lo transfiere a la aplicación llamando al método OnReceiveMessage de la interfaz IApplication.

El campo destAddress de un objeto AdaptationMiddlewareMessage sirve para indicar al protocolo de red cuál es el nodo destino de mensaje, por ende este campo debe ser asignado por la aplicación antes de enviar el mensaje.

Para finalizar el tema cabe mencionar que en la arquitectura no se abordó la cuestión sobre las diferentes representaciones de direcciones de capa de red que pudieran tener sus diferentes tipos de protocolos. Por ejemplo, para un protocolo de red tipo diseminación o *geocast* es adecuado calcular el destino de un mensaje en base al contenido de mensaje o por medio de calcular un área de diseminación, pero no es adecuado utilizar direcciones IP para representar nodos destino. La cuestión anterior se dejó como posible trabajo a futuro. Por el momento la arquitectura utiliza solo la representación de direcciones de capa de red IP, debido a que es bastante conocida y utilizada.

4.10 Resumen del capítulo

En este capítulo se describieron los elementos que componen la arquitectura del *middleware* de adaptación PLUGAM propuesto.

Se presentó el modelo de adaptación que sustenta a la arquitectura, el cual plantea separar la funcionalidad de las soluciones de adaptación de un protocolo de capa de red de la funcionalidad específica para la cual fue creado, y a la vez dividir las soluciones de adaptación en elementos de adaptación implementados en el *middleware* mediante componentes de extensión.

Fueron descritos los pasos a seguir para construir la configuración de una solución de adaptación y se establecieron los puntos del proceso adaptable en los que será ejecutada, los cuales fueron llamados modos de adaptación.

Se introdujo el concepto de *plug-ins* como mecanismo para encapsular los componentes de extensión al *middleware* y se definieron las acciones de adaptación especiales implementadas por el mismo.

Finalmente, se describió la interacción de las aplicaciones vehiculares con el *middleware* y la forma en la cual la aplicación seleccionará los protocolos de capa de red que utilizará durante su ejecución.

Capítulo 5

Definición de usuario, extensiones y clasificación de la arquitectura

En este capítulo se describen algunos temas adicionales sobre la arquitectura PLUGAM. En la primera sección se detalla el perfil del usuario potencial de la arquitectura. Posteriormente se mencionan brevemente algunas posibles extensiones a la arquitectura, que permitirán generar más protocolos adaptativos que los descritos en el capítulo anterior. Se concluye este capítulo describiendo algunas clasificaciones de propuestas de adaptación de la literatura y cómo se ubica la arquitectura propuesta respecto a éstas.

5.1 Definición del usuario del middleware

Un aspecto de este trabajo a dejar claro es quién es el usuario de este *middleware* de adaptación; en esta sección abordaremos ese tema.

Los usuarios a quienes va dirigido el *middleware* de adaptación son los desarrolladores de los sistemas vehiculares de las compañías de automóviles, quienes integran al sistema con capacidades para funcionar en redes VANETs, comunicación 3G, etc. El conductor es el usuario final del sistema vehicular, no del *middleware*.

Este *middleware* es una herramienta de desarrollo que ayudará al fabricante de automóvil a crear aplicaciones vehiculares dinámicas mediante una capa de red adaptativa en el sistema vehicular. A continuación presentaremos una caracterización de los sistemas vehiculares propuestos por los fabricantes de automóviles.

El desarrollo de sistemas vehiculares complejos y con capacidades de comunicación al exterior es una tendencia reciente. En el ámbito del negocio de automóviles son

conocidos en la actualidad como *in-car technology* o sistemas de info-entretenimiento. Algunos ejemplos de estos sistemas son Entune (2011) de Toyota, SYNC (2007) y MyFord Touch (2011) de Ford, R-Link (2011) de Renault, NissanConnectSM (2011) de Nissan y Uconnect (2013) Access de Chrysler. En la figura 19 se muestran algunos de estos sistemas.

Figura 19. Ejemplo de sistemas vehiculares de infoentrenimiento actuales, a) R-Link de Renault, b) Entune de Toyota, c) MyFord Touch de Ford

Los sistemas descritos son todos propietarios y cerrados, no compatibles con los sistemas vehiculares de diferente marca o inclusive entre modelos de coches de la misma marca. Lo anterior provoca que estos sistemas vehiculares tengan cada uno su propia pila de protocolos, los protocolos de red y las aplicaciones que ofrecen; es decir el desarrollador del sistema vehicular es también el desarrollador de las aplicaciones, las cuales son específicas para un sistema vehicular.

La mayoría de estos sistemas de info-entretenimiento actuales cuentan con una pequeña pantalla táctil en el tablero del vehículo, el cual provee comunicación de en-

trada y salida con el usuario. Las especificaciones de *hardware* son similares a sistemas embebidos y teléfonos inteligentes de alta gama. Por el lado del *software*, algunos sistemas cuentan con una versión modificada de Windows embebido, Linux o inclusive Android. Además, estos sistemas pueden ser actualizados por medio de un dispositivo USB flash, ya sea yendo al concesionario o descargándolo de Internet al dispositivo USB.

Recientemente, algunas compañías de automóviles están comenzando a proveer la actualización por el canal de comunicación inalámbrica (se le conoce también como actualización *"over the air"*) del sistema vehicular; por ejemplo el sistema del automóvil Tesla modelo S (Wired (2012)) ofrece actualización de este tipo.

Nuestra propuesta de *middleware* de adaptación se ajusta a estos sistemas vehiculares de los fabricantes, debido a que el *middleware* crea sus propias aplicaciones propietarias en tiempo de construcción, y no es posible agregar nuevas aplicaciones por el conductor que es el usuario final. La única posibilidad de agregar una nueva aplicación o protocolo de comunicación, es mediante actualizaciones de *software* de todo el sistema originados del fabricante. El desarrollador del fabricante, el cual es el usuario del *middleware*, también debe definir las soluciones de adaptación.

Además, antes de liberar la versión final del *software* al usuario final, este debe probar y calibrar debidamente el desempeño de las aplicaciones, el impacto de las soluciones de adaptación, y la posible interferencia entre protocolos de red que compartan un mismo canal de comunicación.

5.1.1 Capacidades de hardware de los actuales y futuros sistemas vehiculares

De las especificaciones de *hardware* utilizado en los sistemas vehiculares actuales (de infoentreteniemiento), se encontró que la capacidad de los procesadores es similar al que utilizan en tabletas y teléfonos móviles de media o baja gama, y de hace dos o tres años. Por ejemplo el MyFordTouch utiliza un procesador ARM Cortex A8 de 600 mHz, 512 MB de RAM y 2 GB NAND flash de memoria ROM (TomHardware1 (2012)); el sistema Uconnect de Chrysler utiliza un procesador single Cortex-A8 core de 720Mhz a 1Ghz, 16 GB de memoria ROM eMMC flash y 512 MB de RAM (TomHardware2 (2013)). El ARM Cortex A8 de 1Ghz se utiliza en el Samsung Galaxy S y Google Nexus S.

Recientemente se han visto tendencias importantes en cuando a homogeneizar el *hardware* y software utilizado en los futuros sistemas vehiculares. Tales como que Intel está empezando a proveer soluciones de *hardware* para estos sistemas en base a su procesador para netbooks Intel Atom (IntelNews (2009)) de un núcleo de 1.3Ghz, 1 GB de memoria RAM DDR2 y 4 Gb de memoria ROM eMMC. Ademas, en 2009 surge la alianza GENIVI que promueve la adopción de tecnologías de sistemas vehiculares open source (Genivi (2009)) y recomienda utilizar *hardware* como los procesadores ARM Cortex A9 (con velocidad máxima de 1.5Ghz y de dos núcleos) y el Intel Atom, ambos procesadores ampliamente utilizados en dispositivos móviles y tabletas de hace dos años (el Cortex 9 es el núcleo del procesador móvil Tegra 2).

El proyecto Tizen (Tizen (2012)), hace una recomendación del *hardware* requerido para implementar su arquitectura de sistema vehicular *"open source"*, indican que debe tener al menos 512 MB de RAM y 1Gb de memoria ROM. Inclusive hay iniciativas que promueven utilizar software *open source* en futuros sistemas vehiculares, basados en el sistema operativo Linux (ComputerWorld (2013), Tizen (2012)).

5.2 Arquitectura extendida de PLUGAM

En esta sección se describen de manera breve algunas ideas que se tuvieron para extender la arquitectura. Las siguientes extensiones permitirían soportar un mayor número de protocolos adaptativos de la literatura. Estas ideas son preliminares, por cuestiones de tiempo no se ahondaron lo suficiente para poder integrarlas a la propuesta de arquitectura PLUGAM.

5.2.1 Elementos de contexto específicos del mensaje

En esta extensión a la arquitectura PLUGAM, la idea fue proponer el uso de elementos de contexto específicos del mensaje. El propósito de este tipo de elementos de contexto es que se calcule su valor actual con base en el contenido de un mensaje o encabezado del protocolo de capa de red, debido a lo anterior solo se puede calcular este valor en modos de adaptación que tengan acceso al mensaje que está siendo procesado, los cuales son el *send*, *receive* y *forward*.

Este tipo de elementos de contexto también puede ser visto como información que viene desde afuera del nodo, algún nodo vecino, o el nodo fuente del mensaje. Ejemplo de este tipo de elementos de contexto puede ser la prioridad del mensaje, la información de posición del último nodo que retransmite el mensaje, el número de saltos del mensaje, etc.

El valor de estos elementos de contexto específicos del mensaje son calculados con base en el análisis del contenido del mensaje o el encabezado del protocolo de capa de red, y en el tiempo que se requieren. En la arquitectura PLUGAM, el *middleware* interpreta el contenido del mensaje o encabezado del protocolo de red como un conjunto de bytes, los cuales no puede descifrar por si mismo. Por lo tanto el *middleware* delega la extracción del valor del elemento del contexto a una entidad que sí es capaz de analizarla.

En el caso de un elemento de contexto calculado con base en el contenido del mensaje, el *middleware* solicita a la aplicación vehicular que creó este mensaje, que obtenga este valor. En el caso de un elemento de contexto calculado con base en el encabezado de un protocolo de capa de red del mensaje, el *middleware* solicita al protocolo de capa de red que obtenga este valor.

Los elementos de contexto calculados con base en un encabezado de protocolo de red solo pueden ser utilizados en los modos *receive* y *forward*; no es posible utilizarlo en el modo *send* debido a que el mensaje todavía no se procesa por el protocolo de red, y por ende no se ha creado un encabezado del protocolo de red.

5.2.2 Soluciones de adaptación críticas en cuanto a valor actual del elemento de contexto

Esta extensión de la arquitectura es útil para el caso de que el usuario requiera soluciones de adaptación críticas, en el sentido de requerir el valor más actualizado de los elementos de contexto para poder funcionar correctamente. La idea es cambiar la precisión en la actualización de los elementos de contexto por tiempo de procesamiento adicional, al proponer un cambio a la definición de la configuración de adaptación.

Este cambio involucra agregar un nuevo campo bandera tipo booleano (llamado "GetInmediateValue") a cada campo de elemento de contexto a la entrada de la configuración de adaptación. Esta bandera se utilizará al momento de la ejecución de una solución de adaptación, justo en el momento en que se obtienen los valores más actuales de los elementos de contexto; si la bandera GetInmediateValue es verdadera entonces el manejador de contexto no recupera el valor más actual de ese elemento de contexto de su estructura interna, sino que lo obtendrá directamente de la fuente de contexto, lo cual requiere mayor tiempo de procesamiento.

5.2.3 Otros modos de adaptación adicionales

La arquitectura PLUGAM se puede extender mediante la introducción de nuevos modos de adaptación. En el transcurso de este trabajo se han detectado dos nuevos modos de adaptación que creemos ampliarían bastaste el conjunto de protocolos adaptativos que se pueden crear utilizando la arquitectura PLUGAM, los cuales se describen a continuación:

- *Modo de adaptación evento de protocolo*, el cual ejecutaría una solución de adaptación cuando un protocolo de capa de red lo indique, informando mediante un evento asíncrono al *middleware*. Para implementar esta idea es necesario modificar la interfaz IProtocolPlugin agregando un mecanismo de publicación y subscripción, de esta manera el protocolo informa el evento al *middleware*. Además se podría definir que un protocolo de red pueda ofrecer uno a varios eventos de protocolo.
- *Modo de adaptación condiciones de contexto*, en el cual la idea es que se ejecute una solución de adaptación al momento que ciertas condiciones se cumplan entre los elementos de contexto del entorno. Por ejemplo, cuando el valor de un elemento de contexto de localización sea tal que indique que está en cierta área. Aquí se puede agregar álgebra booleana para que se tomen en cuenta condiciones complejas constituidas de varios elementos de contexto. Un aspecto a considerar, es que las soluciones de adaptación en este modo deben restringir que solo se utilicen elementos de contexto con tipo de datos conocidos por el *middleware*, además que sean compatibles con las operaciones para evaluar las condiciones complejas (por ejemplo, de tipo de datos booleano, enteros, reales, cadenas de caracteres, etc.)

5.3 Clasificación de la arquitectura PLUGAM

En esta sección se presentan diferentes clasificaciones de propuestas de adaptación, con el objetivo de ubicar a nuestra propuesta de arquitectura de adaptación PLUGAM.

En cuanto a la adaptación de protocolos de comunicación, Grace (2009) distingue entre dos clases de adaptaciones distintas: adaptaciones locales al nodo y adaptaciones distribuidas. Las adaptaciones locales al nodo son iniciadas por un nodo y solo involucra cambiar el comportamiento del mismo. Las adaptaciones distribuidas son iniciadas por un nodo e involucra cambiar el comportamiento del mismo y de otros nodos en la red; un ejemplo es el trabajo sobre reconfiguración dinámica de redes de sensores de Grace (2009).

Las adaptaciones distribuidas son más complejas debido a que se deben considerar aspectos adicionales tales como proponer un mecanismo de consenso para lidiar con que varios nodos realicen una adaptación distribuida con diferentes resultados, debido a contar con diferente información de contexto local. Otro aspecto a considerar en adaptaciones distribuidas, es la necesidad de un protocolo de notificación y calendarización utilizado para realizar una operación de adaptación en todos los nodos participantes. Por último, se tiene que tener cuidado para evitar realizar adaptaciones conflictivas que ocasionen resultados inesperados.

La arquitectura PLUGAM está enfocada a tratar solo adaptaciones locales al nodo, y se deja como trabajo a futuro el extender la arquitectura para dar soporte a adaptaciones distribuidas.

Grace (2009) también identifica dos enfoques generales para la adaptación de *software*: adaptación de parámetros y de composición. La adaptación de parámetros modifica variables predefinidas del programa que determinan el comportamiento del sis-

tema, no existe la introducción de nuevos algoritmos o comportamientos. En cambio la adaptación de composición permite la recomposición dinámica de módulos de software dentro del sistema o introducir nuevo comportamiento. La arquitectura PLUGAM utiliza el enfoque de adaptación de parámetros.

En cuanto a las propuestas de *middleware* de adaptación, Liu y Cheng (2008) las dividen en dos categorías:

- Categoría transparente a la aplicación, en la cual la adaptación ocurre en el *middleware*. Los *middleware* de categoría transparente a la aplicación reducen la complejidad de las aplicaciones, debido a que la información contextual se oculta de ellas y la adaptación es controlada completamente por el *middleware*. Sin embargo, tal *middleware* solo puede proveer adaptación de mejor esfuerzo debido a que las características y objetivos de las aplicaciones son desconocidas por el *middleware*.
- Categoría consciente a la aplicación, en la cual la adaptación ocurre en la aplicación. En los *middleware* de categoría consciente a la aplicación, debido a que la aplicación controla directamente su comportamiento adaptativo, esto facilita la reutilización del *middleware* para construir aplicaciones adaptativas.

La arquitectura PLUGAM corresponde a la categoría de transparente a la aplicación, debido a que el *middleware* se encarga de ejecutar las soluciones de adaptación de manera automática y transparente a las aplicaciones vehiculares.

5.4 Resumen del capítulo

Este capítulo describe algunos elementos que permiten extender la funcionalidad de la arquitectura del *middleware* de adaptación PLUGAM, como los elementos de contexto específicos del mensaje, soluciones de adaptación críticas en cuanto al valor actual del

elemento de contexto, y otros modos de adaptación.

La arquitectura *middleware* fue descrita como una herramienta que provee de una capa de red adaptativa a las aplicaciones vehiculares, se habló de los usuarios a quienes va dirigida la arquitectura del *middleware* y sobre la tendencia actual de las compañías fabricantes de automóviles a desarrollar sistemas vehiculares con capacidades de comunicación al exterior. Adicionalmente se presentaron algunos sistemas vehiculares de tipo info-entretenimiento desarrollados por fabricantes de automóviles.

Finalmente, se presentó la clasificación de las propuestas de adaptación encontradas en la literatura con la finalidad de compararlas con el *middleware* de adaptación PLUGAM.

En el siguiente capítulo se presentarán los casos de estudio que se realizaron en este trabajo.

Capítulo 6

Descripción de casos de estudio

En este capítulo se presentan dos casos de estudio de protocolos adaptativos, los cuales son construidos con nuestra arquitectura de *middleware* de adaptación, y además cada uno trata un comportamiento distintivo del *middleware*. Estos casos de estudio nos ayudan a explicar cómo funcionan en conjunto todos los aspectos de la arquitectura, además que sirven como prueba de concepto de la misma.

El primer caso de estudio trata sobre extender un protocolo de red MANET existente agregando una adaptación con base en contexto local, para de esta manera hacerlo más apto de ser utilizado en redes VANET. El segundo caso de estudio propone implementar un escenario hipotético utilizando la arquitectura PLUGAM, en este escenario se realiza una selección o cambio entre dos protocolos de red.

A continuación se presenta una descripción detallada de estos casos de estudios y cómo estos se implementan en la arquitectura PLUGAM.

6.1 Caso de estudio 1: Adaptación interna del protocolo de red con base en información de contexto

En este caso de estudio el objetivo es extender un protocolo de red ya definido mediante la adición de funcionalidad de adaptación a cierta información de contexto. Como protocolo de red base se seleccionó a OLSR, el cual es un protocolo de MANETs. Se seleccionó a uno de MANET debido a que todavía no existen protocolos de referencia VANETs, en cambio en MANET si los hay, siendo OLSR uno de ellos. Aparte, OLSR es un protocolo particularmente bien conocido, con mucho trabajo a su alrededor y con implementaciones disponibles al público.

OLSR es un protocolo de red proactivo debido a que mantiene actualizada una tabla de enrutamiento, lo anterior mediante el uso de mensajes de control periódicos (por ejemplo los mensajes de control HELLO) recibidos por sus vecinos.

El trabajo de Huang *et al.* (2006) experimenta con la constante HELLO_INTERVAL de OLSR, la cual indica el periodo de tiempo en el que se envían estos mensajes HELLO, esta constante está fuertemente relacionada con el hecho de tener lo más actualizada posible la tabla de enrutamiento. Los autores midieron el desempeño del protocolo cambiando el valor de esta constante en diferentes escenarios de densidad y movilidad de nodos. Concluyeron que ajustar esta constante en escenarios específicos puede generar mejoras en el desempeño, además proponen como trabajo a futuro adaptar en tiempo de ejecución esta constante.

La propuesta de adaptación de este caso de estudio 1 toma como base los resultados y trabajo a futuro de Huang *et al.* (2006), de esta manera se tiene la certeza que el impacto de la propuesta de adaptación traerá en realidad un beneficio positivo.

Haciendo una analogía con los conceptos de la arquitectura de adaptación PLUGAM, la constante (ahora variable) HELLO_INTERVAL es representada como una acción de adaptación ofrecida por el *plug-in* tipo protocolo encapsulando a OLSR. Los elementos de contexto, los cuales son la entrada de la adaptación, serán representados por un solo elemento de contexto asociado a la velocidad del nodo/vehículo, lo cual es una simplificación de la información de contexto sobre la movilidad del nodo. Existen otras maneras más complejas de representar la movilidad de nodo, por ejemplo en Boleng *et al.* (2002) la métrica de movilidad está basada en la duración del enlace entre nodos vecinos; sin embargo, con motivo de describir un caso de estudio simple, utilizar la velocidad del vehículo como simplificación de la movilidad del nodo cumple con este propósito.

Además, en este caso de estudio se propone obtener la velocidad del vehículo creando un *plug-in* tipo ContextE, el cual encapsula al código para procesar las lecturas de un GPS, en lo cual está la velocidad. A su vez, estos valores de velocidad son calculados dentro del GPS midiendo el desplazamiento *"Doppler"* entre las señales de los satélites. A nivel de la arquitectura, el *plug-in* ContextE que encapsula al GPS ofrecerá al *middleware* de adaptación el elemento de contexto "velocidad del vehículo".

En cuanto al algoritmo de adaptación propuesto para este caso de estudio, se decidió proponer un algoritmo simple, basado en una asignación uno-a-uno entre la velocidad del vehículo y un valor de HELLO_INTERVAL. La relación entre éste par de valores sigue un razonamiento simple: a velocidades altas los nodos vecinos cambian más frecuentemente, por ende la tabla de vecinos del nodo debe ser actualizada más seguido bajando el valor del HELLO_INTERVAL. Este razonamiento no toma en cuenta la existencia de nodos vecinos manejando también a altas velocidades en la misma dirección del vehículo. Sin embargo, con el propósito de ejemplificar un caso de estudio sencillo este razonamiento es suficiente.

Para implementar la asignación uno-a-uno entre velocidad y HELLO_INTERVAL, se definieron los pares $(v_i, f(v_i))$, en donde $i \in \{1, ..., N\} \subseteq \aleph$, v_i representa la velocidad del vehículo en km/h, $f(v_i)$ representa el valor futuro del HELLO_INTERVAL en milisegundos (2000 milisegundos es el valor por omisión utilizado por OLSR). Para este algoritmo en particular se seleccionaron $N = 7$ pares, los cuales se muestran en la Tabla 2.

Utilizando los pares $(v_i, f(v_i))$ se obtiene el HELLO_INTERVAL $f(v)$, donde $v \in \Re$, de la siguiente manera:

Tabla 2. Tabla mostrando los pares $(v_i, f(v_i))$ propuestos para el algoritmo de adaptación del caso de estudio 1

v_i	$f(v_i)$
0	3500
10	3400
20	3000
30	2000
70	1000
90	900
120	800

$$f(v) = InterpolacionLineal(v_i, v_{i+1}) \mid v \in [v_i, v_{i+1}) \ , \ i \in [1, N-1]$$

$$Si\ v < v_1\ entonces\ f(v) = f(v_1)$$

$$Si\ v > v_N\ entonces\ f(v) = f(v_N)$$

Finalmente, la solución de adaptación propuesta para el caso de estudio es puesta en el modo de adaptación CUPIB. En este modo la solución de adaptación se ejecuta periódicamente, por consiguiente se actualiza el valor del HELLO_INTERVAL. Se definió el periodo del CUPIB en 1000 ms (mediante el campo CUPIBTime de la configuración de adaptación), debido a que es la tasa de actualización de datos de un dispositivo GPS cualquiera.

Ahora se procede a describir cómo se construye el caso de estudio 1 en la arquitectura PLUGAM. El primer paso es construir los componentes de extensión que se necesitan, mediante la construcción de los siguientes *plug-ins* (ver la figura 20):

- Crear un *plug-in* tipo protocolo de OLSR, este *plug-in* ofrece una acción de adaptación ligada a la variable (antes constante) HELLO_INTERVAL de OLSR.
- Construir un *plug-in* tipo contextE, el cual ofrece el elemento de contexto velocidad del vehículo. El *plug-in* obtiene este contexto encapsulando al código para

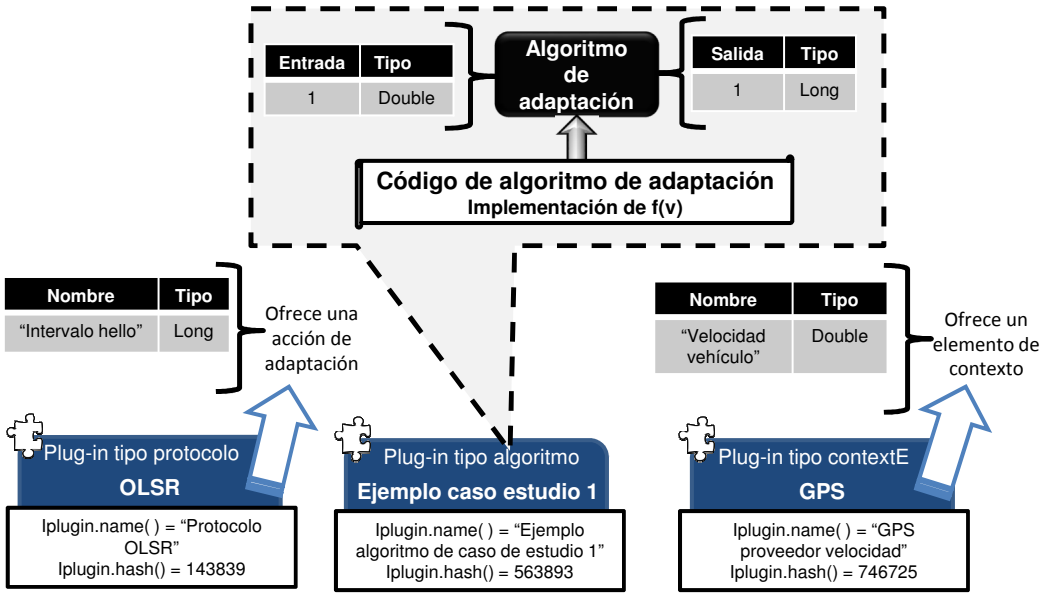

Figura 20. Componentes de extensión necesarios para implementación del caso de estudio 1 en el middleware de adaptación PLUGAM

recuperar la información de un dispositivo GPS.

- Crear un *plug-in* tipo algoritmo que computa a $f(v)$. Este algoritmo recibe como entrada un tipo de dato real (Double en Java) conteniendo el valor de velocidad del vehículo, y como salida un elemento de tipo entero (Long en Java) relacionado al valor del HELLO_INTERVAL.

Ya que se tienen definidos los componentes de extensión necesarios para el caso de estudio 1, el siguiente paso es ligarlos entre ellos mediante la declaración de una configuración de adaptación, tal y como se muestra en la figura 21. La configuración mostrada aquí está en formato JSON (Crockford (2006)), debido a que así se implementó en el prototipo del *middleware*. Observando el código JSON se puede observar que los elementos que constituyen la configuración de adaptación son objetos tipo ElementIdentifier representados en JSON como la cadena de caracteres {"name":, "hash":, "secondname":}.

Además, se observa que esta configuración de adaptación se define en el modo CUPIB y se asigna 1000 al campo CUPIBTime.

Declaración de una configuración de adaptación en formato JSON


```
[ {
   "configurationName": "Configuración de caso de estudio 1",
   "algorithm": { "hash": 563893, "name": "Ejemplo algoritmo de caso de estudio 1", "secondName": "" },
   "contextElements": [ { "hash": 746725, "name": "GPS proveedor velocidad", "secondName": "Velocidad vehículo" } ],
   "actionElements": [ { "hash": 143839, "name": "Protocolo OLSR", "secondName": "Intervalo hello" } ]
   "adaptationMode": "CUPIB",
   "CUPIBTime": 1000,
   "protocol": { "hash": 143839, "name": "Protocolo OLSR", "secondName": "" },
   "application": { "hash": 0, "name": "", "secondName": "" },
} ]
```


Figura 21. Configuración de adaptación del caso de estudio 1 en formato JSON

Lo anterior es todo lo que se necesita para recrear la adaptación del caso de estudio 1 en el *middleware* PLUGAM. En tiempo de ejecución la solución de adaptación es procesada periódicamente debido al funcionamiento del modo de adaptación CUPIB, por consiguiente cambiando el valor del HELLO_INTERVAL de OLSR dependiendo del valor de la velocidad del vehículo.

6.2 Caso de estudio 2: Cambio de protocolos con base en información de contexto

En este caso de estudio se presenta una adaptación donde se realiza una selección o cambio entre diferentes protocolos de red, con base en el contexto actual. El caso de estudio que se describe a continuación es un escenario de concepto simple donde se necesita una adaptación de este tipo.

Este escenario se centra en una aplicación vehicular encargada de diseminar un evento de accidente a los vehículos interesados manejando en una carretera. Se estableció que esta aplicación utiliza un protocolo de capa de red tipo *broadcast* direccional (Shen *et al.* (2006)), con el objetivo de limitar la diseminación del mensaje solo hacia los vehículos moviéndose en la misma dirección del vehículo involucrado en el accidente (el cual es el generador del evento).

El escenario anterior funciona correctamente mientras existan suficientes vehículos circulando en la misma dirección (escenario A de la figura 22). Sin embargo, si no existen suficientes vehículos en la misma dirección que retransmitan el mensaje, este no continúa su recorrido. Para solucionar este fallo en el escenario, se propone aprovechar el movimiento de los vehículos circulando en la dirección contraria de la carretera.

Más específicamente, dado este contexto del fallo se propone cambiar e utilizar un protocolo *carry and forward* (escenario B en la figura 22). Las condiciones para realizar el cambio de protocolos son las siguientes: si no existen vehículos vecinos moviéndose en la misma dirección y si se detecta un vehículo vecino pasando en la dirección opuesta. Cabe resaltar que esta condición de cambio se evalúa independientemente para cada mensaje. En caso que se cumplan estas condiciones, para ese mensaje en particular, se cambia la utilización del protocolo de *broadcast* direccional por el *carry and forward*. En caso que no se cumplan las condiciones, entonces se continúa utilizando el protocolo *broadcast* direccional.

En cuanto a la solución de adaptación de este escenario, tiene sentido poner una solución ya sea en el modo de adaptación send y/o forward; dependiendo de qué modo se

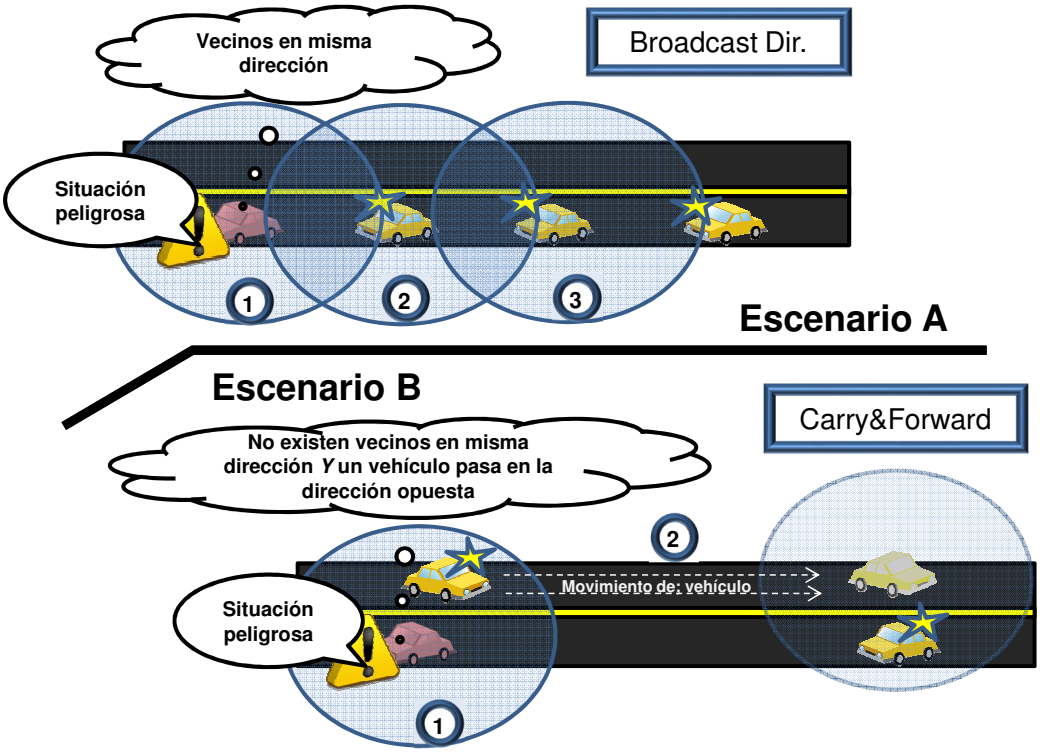

Figura 22. Escenario del caso de estudio 2 el cual propone un cambio entre dos protocolos de capa de red

utiliza, el comportamiento es diferente. En caso que se coloque la solución de adaptación en el modo send, entonces el cambio entre protocolos se da solo en el nodo fuente; específicamente cuando la aplicación vehicular llama al comando "enviar mensaje" en el middleware eligiendo al protocolo de red. Por otro lado, si se coloca la solución de adaptación en el modo forward, esta se ejecutará en cada salto o nodo retransmisor del mensaje. Cabe mencionar que es posible obtener estos dos comportamientos, solo se requiere definir dos configuraciones de adaptación pero colocadas cada una en un modo de adaptación.

Una vez descrito el escenario del caso de estudio 2, a continuación se muestra como se construye en la arquitectura PLUGAM. En primer lugar se describen los plug-ins que

proveerán los componentes de extensión necesarios para esta adaptación, estos plug-ins se muestran en la figura 23 y son los siguientes:

- Dos plug-ins tipo protocolo, uno implementando un protocolo de red broadcast direccional, el otro encapsulando a un protocolo tipo carry and forward. Por otro lado, se decidió que los elementos de contexto necesarios para esta solución de adaptación las proporcione el protocolo carry and forward, estos elementos de contexto serán un valor booleano indicando si está o no está pasando un vehículo en dirección contraria, y otro valor booleano indicando si existen o no vehículos vecinos conduciendo en la misma dirección. Cabe resaltar que el cómputo de los valores de estos elementos de contexto no es trivial; el cómo realizarlos se delega a la implementación del protocolo y es transparente para la solución de adaptación, es por eso que este detalle se omite en la descripción del caso de estudio.
- Además se necesita un plug-in tipo aplicación vehicular conteniendo a la aplicación evento de accidente en el camino. Esta requiere para su funcionamiento interno los protocolos de red broadcast direccional y el carry and forward, por ende sus objetos ElementIdentifier deben ser incluídos en la lista que regresa el método GetAppDeclaredProtocols de la interfaz del IApplicationPlugin del plug-in tipo aplicación.
- Finalmente, se requiere un plug-in tipo algoritmo conteniendo la lógica a seguir para seleccionar el protocolo que se va a utilizar, entre los dos disponibles. El tipo de dato de la salida de este algoritmo debe ser igual a la entrada de la acción de adaptación especial switcher, la cual es de tipo ElementIdentifier ya que hace referencia a un componente de extensión protocolo de red. Internamente el algoritmo contiene una simple operación condicional utilizando los valores booleanos

de los elementos de contexto recibidos de entrada.

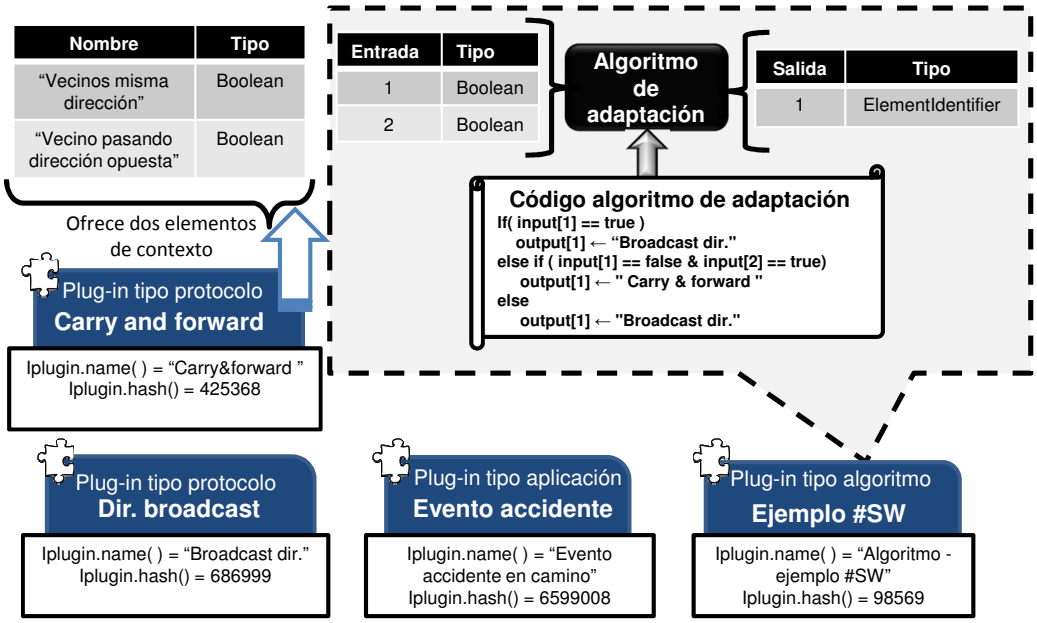

Figura 23. Componentes de extensión/plug-ins necesarios para la implementación del caso de estudio 2 en la arquitectura PLUGAM

El siguiente paso en la construcción del caso de estudio es el ligar la funcionalidad de los componentes de extensión, por medio de la declaración de una configuración de adaptación. La figura 24 muestra esta configuración expresada de nuevo en formato JSON. Una observación importante sobre una configuración declarada en el modo de adaptación *send*, es que es obligatorio asignar un valor a los campos "application" y "protocol". Lo anterior debido a que estos campos permiten limitar que solo se ejecute la solución de adaptación cuando la aplicación especificada (en este caso la aplicación evento de accidente en el camino) manda un mensaje al protocolo especificado (en este caso el protocolo de red *broadcast* direccional). El protocolo especificado en la configuración también actúa como el protocolo por omisión del escenario del caso de estudio.

Declaración de una configuración de adaptación en formato JSON

```
[ {
   "configurationName": "Conf caso estudio 2 #SW",
   "algorithm": { "hash": 98569, "name": "Algoritmo - ejemplo #SW", "secondName": "" },
   "contextElements": [ { "hash": 425368, "name": "Carry&forward", "secondName": "Vecinos misma dirección" },
                        { "hash": 425368, "name": "Carry&forward", "secondName": "Vecino pasando dirección opuesta" } ],
   "actionElements": [ { "hash": 0, "name": "#SPECIAL#", "secondName": "#SpecialASwitcher#" } ]
   "adaptationMode": "Send",
   "CUPIBTime": 0,
   "protocol": { "hash": 686999, "name": "Broadcast dir.", "secondName": "" },
   "application": { "hash": 6599008, "name": "Evento accidente en camino", "secondName": "" },
} ]
```

Figura 24. Configuración de adaptación para el caso de estudio 2 en formato JSON

Después de agregar los componentes de extensión necesarios y crear la configuración de adaptación, ya en tiempo de ejecución el *middleware* procesa la solución de adaptación en el momento indicado y automáticamente. Para finalizar la descripción de este caso de estudio, a continuación se muestra cómo se realiza este tipo de adaptación adentro del *middleware* en tiempo de ejecución (ver figura 25).

El proceso de adaptación de la figura 25 comienza cuando la aplicación evento de accidente en el camino llama al comando *send*, especificando que el mensaje se disemine utilizando el protocolo de red *broadcast* direccional (1). En vez de irse directamente al protocolo de red, el mensaje es interceptado por el *middleware* y lo manda al módulo que ejecuta una solución de adaptación en el modo *send*. En este módulo se realiza una búsqueda de una configuración que está definida en el modo *send* y para este par aplicación-protocolo (2), lo cual en este ejemplo sí existe y tiene el nombre "Conf caso estudio 2 #SW".

El siguiente paso en este proceso es la ejecución de la solución de adaptación definida en la configuración (3). Primeramente el *middleware* obtiene todos los valores actuales

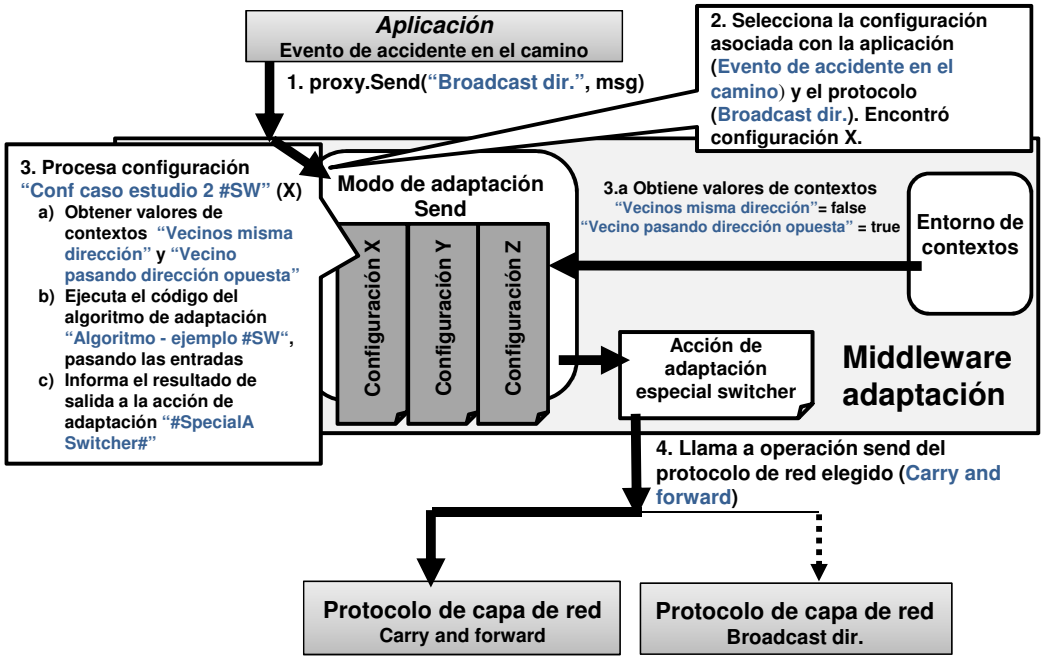

Figura 25. Descripción gráfica de comportamiento en tiempo de ejecución del middleware procesando la solución de adaptación del caso de estudio 2

de los elementos de contexto con ayuda del manejador de contextos (3.a). Después estos valores son pasados como entrada del algoritmo de adaptación, se ejecuta el algoritmo y este produce el valor de salida que está asociado a la acción de adaptación especial *switcher*.

Como último paso el *middleware* transfiere el valor de salida del algoritmo al módulo *switcher*, el cual es el responsable de orientar el mensaje al protocolo de red que resultó elegido (4). En el ejemplo de ejecución de la figura 25, los valores actuales de los elementos de contexto son tales que ocasionan que la adaptación direccione el mensaje al protocolo *carry and forward*.

6.3 Resumen del capítulo

Este capítulo presentó dos casos de estudio realizados para mostrar el funcionamiento de la arquitectura del *middleware* de adaptación PLUGAM.

El primer caso de estudio mostró cómo se agrega la funcionalidad de adaptación a un protocolo de capa de red predefinido, tomando como entrada la información del contexto.

El segundo caso de estudio mostró la selección del protocolo de capa de red que realiza el *middleware* con base a la información de contexto.

En el siguiente capítulo se presentará la evaluación realizada al prototipo del *middleware* de adaptación y el análisis de resultados.

Capítulo 7

Evaluación

La evaluación de este trabajo se divide en tres fases. Primeramente se presenta una evaluación de costo/desempeño, la cual está basada en un conjunto de mediciones sobre el costo de procesamiento y de memoria adicional hechas con respecto a un prototipo (la descripción del prototipo se encuentra en el apéndice A).

En la fase dos de la evaluación se presenta una simulación de uno de los casos de estudio descritos en el capítulo 6, esto con el fin de analizar el impacto que tiene la adaptación propuesta en el caso de estudio e ir más allá de solo lo teórico.

Finalmente, en la fase 3 se discute la escalabilidad de la arquitectura PLUGAM, además se describen los costos adicionales introducidos por este enfoque genérico de la arquitectura, en contraste con los protocolos adaptativos existentes, los cuales son soluciones particulares.

7.1 Evaluación de costo/desempeño

Esta fase de la evaluación consta de una serie de mediciones hechas al prototipo. El objetivo de estas mediciones es determinar la viabilidad de implementar en un sistema vehicular la arquitectura del *middleware* propuesto. De manera concisa, se realizaron mediciones a los siguientes aspectos del prototipo:

- Requerimiento en memoria.
- Costos de procesamiento o latencia adicional generada por el middleware al ejecutar la solución de adaptación en los diferentes modos de adaptación.
- Costo en memoria y procesamiento introducido por la plataforma de plug-ins propuesta.

Las mediciones al prototipo fueron realizadas en una computadora MacBook Air modelo A1231 (2008), la cual contiene un procesador Intel Core Duo 2 de 1.6Ghz y 2 GB de memoria RAM. En el lado del *software*, se trabajó con el sistema operativo Ubuntu 10.10, la máquina virtual de Java usada fue la JRE versión 1.6.0.22. Además, se utilizó el contenedor OSGi Apache Felix versión 4.02 e iPOJO versión 1.8.2. Se optó utilizar un *hardware* Mac debido a que es que es más fácil replicar los resultados, ya que todas las computadoras de uno de sus modelo contienen exactamente los mismos componentes internos.

Se eligieron estas especificaciones de *hardware* y *software* debido a que probablemente serán de capacidad similar a los sistemas vehiculares de los próximos años. Aunque lo ideal hubiera sido realizar las mediciones de tiempo de procesamiento utilizando alguno de los sistemas vehiculares que existen actualmente (info-entreteniemiento), pero estos son sistemas propietarios y cerrados por lo que no se tuvo acceso a ellos para hacer los experimentos en escenarios reales. Sin embargo, aunque es difícil obtener las capacidades de cómputo exactas, aun así existe información en la web y tendencias del mercado (esto se describió en la sección 5.1) que nos permite inferir las capacidades de cómputo en un futuro cercano y mediano plazo de estos sistemas.

Por otro lado, todas las mediciones fueron repetidas 100 veces y lo que se muestra son los valores promedio y los intervalos de confianza de estas mediciones, con un nivel de confianza de 95%.
A continuación se describe cómo se realizaron estas mediciones y se muestran los resultados obtenidos.

7.1.1 Mediciones sobre los requerimiento en memoria del prototipo

Debido al interés y posibilidad de implementar estos sistemas vehiculares en sistemas embebidos o en *hardware*/*software* similar al de los teléfonos inteligentes (los cuales

tienen un limitado poder de procesamiento y espacio de memoria), resulta prudente tener una idea de los costos de memoria que genera nuestra propuesta de *middleware* de adaptación, especialmente el uso de memoria RAM y ROM.

La cantidad de memoria RAM utilizada se obtuvo observando el consumo de memoria del proceso de "java" (el proceso de la máquina virtual) desde el programa "ps" en Linux. Con respecto al valor de ROM, éste se obtuvo del consumo en disco duro que tienen los archivos binarios requeridos para correr el prototipo; es decir la máquina virtual de java, el contenedor OSGi y los archivos de iPOJO, así como los archivos binarios del *middleware* de adaptación.

Los requerimientos generales de memoria obtenidos, son los siguientes:

- 20.1 MB de memoria RAM.
- 111.9 MB de ROM.

Ahondando en los 20.1 MB de RAM requeridos (ver figura 26), se observa que 16.7 MB corresponden al entorno de la plataforma de *plug-in* (JVM + Apache Felix + iPOJO). Además, solamente la máquina virtual de Java consume 6.72 de estos 20.1 MB.

En cuanto a los 111.9 MB de ROM requeridos (ver figura 26), 109.5 MB corresponden a los archivos de la máquina virtual de Java. Los archivos binarios del contenedor OSGi y de iPOJO ocupan 1.98 MB de ROM, y los restantes 442 KB corresponden a los archivos del *middleware* de adaptación. En la figura 26 se incluyen el espacio en ROM que ocuparon los *plug-ins* que implementan el caso de estudio 1 en el prototipo, el cual fue de 4.03 MB; la mayor parte de estos 4 MB de ROM provienen del código del protocolo de red OLSR, debido a que éste se implementó utilizando unas librerías de *software* complejas.

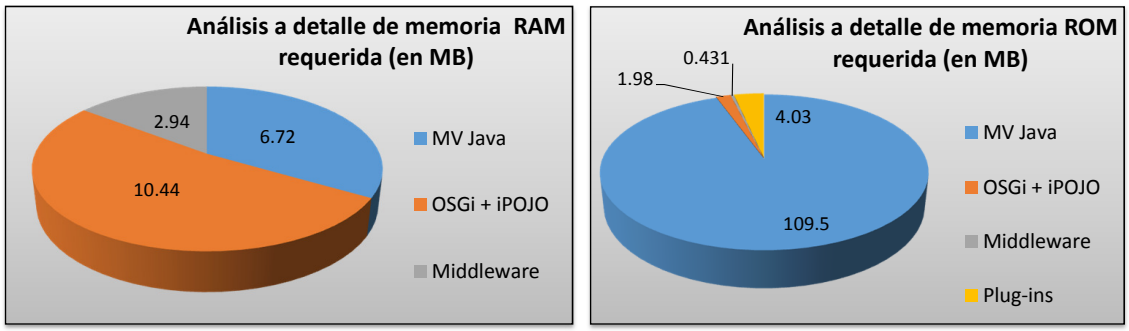

Figura 26. Análisis del requerimiento de memoria RAM y ROM del prototipo

Un costo adicional de memoria importante en la arquitectura PLUGAM es el uso de un encabezado de control adicional, el cual se agrega al contenido del mensaje de una aplicación y contiene información útil para el funcionamiento del *middleware*. En particular, este encabezado contiene los campos del objeto AdaptationMiddlewareMessage (mostrado en la figura 18).

Debido a que los campos del AdaptationMiddlewareMessage constan de varios objetos ElementIdentifier, los cuales a su vez tienen cadenas de caracteres, el tamaño máximo de las cadenas influye fuertemente en el tamaño en bytes del encabezado. Fijando el tamaño máximo de las cadenas a 30 caracteres (en formato ANSI de un byte cada carácter), además de utilizar una representación de direcciones de red IP de 4 bytes, da como resultado un tamaño de encabezado de 132 bytes.

En resumen, los requerimientos ROM y RAM obtenidos del prototipo son modestos, inclusive compatibles con los requerimientos de aplicaciones para dispositivos móviles con sistema operativo Android (2007); los cuales tienen disponible cuando menos 512 MB de memoria ROM y 512 MB de RAM. Se compara con dispositivos móviles con Android debido a que es relativamente sencillo implementar la plataforma de plug-ins del prototipo y en consecuencia todo el middleware en este sistema operativo.

Por otro lado, se encontró que la máquina virtual de Java (JVM) es la responsable

de la mayor utilización de memoria ROM del prototipo. En cuanto a memoria RAM, la plataforma de *plug-ins* basada en Apache Felix utiliza la mayor parte de esta memoria. Por último, el tamaño del encabezado del *middleware* es grande en comparación con otros encabezados de protocolos de red y sistemas *middleware*. Sin embargo, éste puede ser reducido cambiando la manera en que se hace referencia a los componentes de extensión, es decir redefiniendo los objetos ElementIdentifier, preferentemente omitiendo el uso de cadenas de caracteres como manera de identificación.

7.1.2 Mediciones sobre costo en procesamiento del prototipo

El objetivo de obtener estas mediciones, es tener un valor de referencia sobre el costo de procesamiento o latencia introducido por el *middleware* de adaptación. Existen varias maneras de medir el costo de procesamiento; por ejemplo, la cantidad de operaciones, los accesos a memoria, el tiempo de procesamiento, entre otras. En este trabajo se asocia el costo de procesamiento al tiempo de procesamiento requerido para ejecutar algún proceso del *middleware* de adaptación, por consiguiente estos dos conceptos los utilizaremos como sinónimos a partir de ahora.

En los modos *send*, *receive* y *forward*, existe un costo de procesamiento adicional debido a que el *middleware* intercepta el paso de mensajes entre aplicaciones y los protocolos de red, con el fin de ejecutar una solución de adaptación en este punto. En el caso de modo CUPIB, el tiempo requerido para calcular de nuevo las variables internas de los protocolos de red depende del costo de procesamiento asociado a ejecutar una solución de adaptación en este punto.

En particular, es de interés medir el tiempo o latencia que tarda en ejecutar una solución de adaptación en los cuatro modos de adaptación y con diferentes variantes. Entre estas variantes están medir con diferentes algoritmos de adaptación, cantidad de

elementos de contexto de entrada y cantidad de acciones de adaptación como salida.

Descripción del marco de trabajo para las mediciones de costo de procesamiento

A continuación se describe el marco de trabajo utilizado para realizar las mediciones de tiempo de procesamiento.

Debido a que los algoritmos de adaptación son definidos por el desarrollador del *plug-in*, estos pueden tener cualquier complejidad, cantidad y tipo de instrucciones u operaciones. Para estas mediciones de costo de procesamiento, se definieron cuatro perfiles de un algoritmo de adaptación, los cuales son: nulo, bajo, medio y alto.

La utilidad de estos perfiles es indicar de manera abstracta qué tan complicada y tardadas son las operaciones internas del algoritmo. Un algoritmo de adaptación tal como lo definimos, solo no contiene operaciones de comunicación tales como envió o recepción de mensajes, ni siquiera operaciones de acceso a disco, más bien son instrucciones tales como aritméticas, comparaciones, operaciones matemáticas más complejas y ciclos. Es por eso que se consideró representar su complejidad en base a la cantidad operaciones aritméticas que contiene el algoritmo, se eligió tomar en cuenta las operaciones de punto flotante por requerir mayor capacidad de procesamiento. A continuación se describe cada uno de estos perfiles o carga de algoritmo:

- *Nulo:* El algoritmo de adaptación no tiene ninguna instrucción de punto flotante, no se realiza tarea alguna.
- *Bajo:* En este perfil se intenta caracterizar a aquellos algoritmos de adaptación que hagan solo algunas asignaciones de variables, operaciones de comparaciones o matemáticas con datos de entrada (que son los valores de los elementos de contexto). Se representa por un algoritmo de costo $O(n)$, específicamente se

utilizó un algoritmo que calcula el promedio de una lista de n números reales.

- *Medio:* La intención es que en este perfil se encuentren aquellos algoritmos que implementan operaciones o instrucciones más sofisticadas con los datos de entrada. Se representa con un algoritmo de costo $O(n^2)$, específicamente se utilizó el algoritmo de ordenamiento por selección, donde se ordenan n números reales.
- *Alto:* Un algoritmo de adaptación con este perfil es aquel que tiene una cantidad alta de operaciones e instrucciones complejas, se considera un caso extremo debido a que incrementaría notablemente el retraso de procesamiento o *"processing delay"* (Kurose y Ross (2009)) del *middleware* y por consiguiente del nodo de red. Se representa por un algoritmo de costo $O(\log(n)n^2)$, específicamente se utilizó un algoritmo que ordena n números reales $\text{Log}(n)$ veces, utilizando el algoritmo de ordenamiento por selección.

Adicionalmente, el concepto de carga de un algoritmo debe cambiar dependiendo de la cantidad de elementos de contexto (NUM_ECONTEXTO) de entrada al algoritmo y del número de acciones de adaptación de salida del mismo (NUM_ACCIONESA). Para lograr lo anterior, redefinimos a n, que en los perfiles anteriores representaba el tamaño de las lista de números reales, como lo indica la equación 2, donde $m = 100$ ahora representa el tamaño de la lista de números reales de punto flotante.

$$n = m + \frac{m}{10}(\text{NUM\_ECONTEXTO} + \text{NUM\_ACCIONESA}) \tag{2}$$

Las mediciones de tiempo de procesamiento se muestran en la figura 27. Cada punto de medición es el promedio de correr la misma configuración 100 veces, las gráficas incluyen el intervalo de confianza con un nivel de 95%. Las gráficas a) y b) muestran los tiempos de procesamiento de los modos de adaptación *send*, *receive* y *forward*, utilizando uno u otro de las acciones de adaptación especiales *filter* y *switcher*. Las abreviaciones SF, SS, RF, RS, FF y FS en el eje horizontal indican precisamente esta combinación entre modos de adaptación y acción de adaptación especial. La primera letra se refiere al modo de adaptación (***s****end*, ***r****eceive*, ***f****orward)*, y la segunda se refiere a la

acción de adaptación especial (***f****ilter*, ***s****witcher*) asociada como única salida al algoritmo.

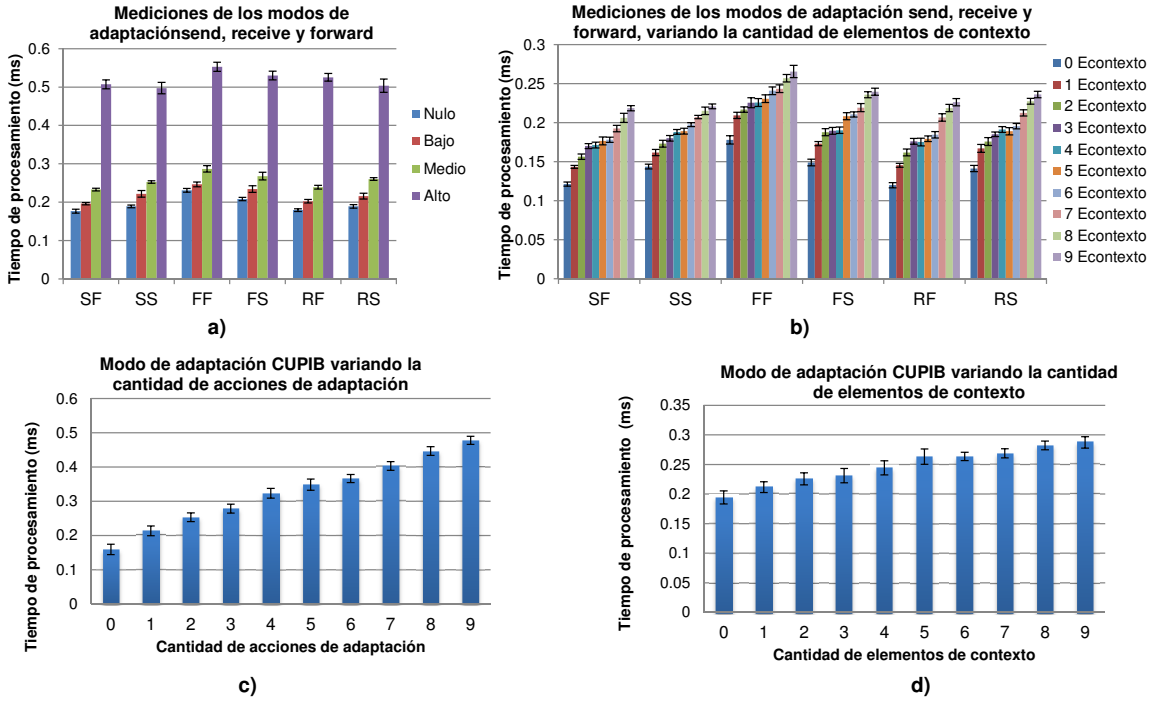

Figura 27. Resultados de las mediciones sobre el costo de procesamiento del prototipo del middleware de adaptación

En a), se muestran estos tiempos pero en las cuatro diferentes cargas de algoritmo de adaptación y fijando la cantidad de elementos de contexto a 5. En b) se varía la cantidad de elementos de contexto o entradas del algoritmo de 0 a 9 y se utiliza la carga del algoritmo nulo.

Por otro lado, las gráficas c) y d) de la figura 27 muestran los tiempos de procesamiento solo del modo de adaptación CUPIB. En la gráfica c), se varía de 0 a 9 la cantidad de acciones de adaptación (tipo protocolo) que se asocian a la salida del algoritmo de adaptación, mientras que se deja fijo en 1 la cantidad de elementos de contexto y se utiliza una carga de algoritmo nulo. En la gráfica d), es al contrario, se varía la cantidad de elementos de contexto mientras se dejan fijas las acciones de adaptación.

Análisis de resultados

Examinando las mediciones anteriores de costo de procesamiento, se observa lo siguiente:

- Todos los tiempos de procesamiento obtenidos para los cuatro modos de adaptación están en el orden de décimas de milisegundos, van desde una décima de milisegundo hasta máximo 1.6 milisegundos.
- El costo de procesamiento más alto obtenido con nuestro marco de trabajo fue de 1.6 ms. Éste se dio en la medición del modo de adaptación CUPIB, con una carga de algoritmo alta, 9 elementos de contexto y 9 acciones de adaptación como entrada y salida del algoritmo, respectivamente. La medición anterior no aparece en las gráficas de la figura 27, debido a que resultó problemático presentarla junto a los datos que sí se graficaron.
- Los modos de adaptación *send* y *receive* producen tiempos de procesamiento similares, esto ocurre debido a que el funcionamiento sobre interceptar el mensaje entre aplicaciones y protocolos de red es parecido en los dos modos, solo el sentido de la interrupción cambia. Después el modo *forward* genera costo de procesamiento un poco más elevado (0.04 ms adicional) a los primeros dos. El modo de adaptación que genera un mayor costo de procesamiento resultó ser el modo CUPIB, aunque por un pequeño margen de diferencia pequeño.
- La mayoría de los incrementos en tiempo de procesamiento al pasar de una carga de algoritmo menor a otra mayor son menores. Sin embargo, pasar de carga de algoritmo medio hacia alta, genera un incremento considerable, de varias décimas de milisegundos en costo de procesamiento. Inclusive genera el caso extremo y no deseable de hacer que el tiempo de procesamiento dentro algoritmo de adaptación sea mayor al resto del funcionamiento del *middleware*.
- Elevar la cantidad de elementos de entrada (elementos de contexto) en el algoritmo, influye poco en el costo de procesamiento obtenido, solo se incrementa unas pocas centésimas de milisegundos por elemento de contexto adicional. Por

el otro lado, elevar la cantidad de elementos de salida del algoritmo (acciones de adaptación), influye más en el costo de procesamiento comparado que los elementos de contexto. El comportamiento anterior se debe a que cada acción de adaptación requiere que el m*iddleware* comunique el valor resultante directamente a la entidad fuente de esta acción de adaptación, esto genera tiempos de procesamiento adicionales. En cambio, la arquitectura PLUGAM se diseñó para que el obtener el valor de un elemento de contexto sea una operación de rápido acceso, evitando que se le pregunte directamente a la entidad fuente del contexto.

Utilizando el marco de trabajo anterior, también se calculó el costo de procesamiento asociado a la ejecución de la solución de adaptación del caso de estudio 1 (descrito en el capítulo 6). Esta solución de adaptación requiere de un solo elemento de contexto (la velocidad del vehículo), está asociado a una acción de adaptación (cambiar el valor del intervalo del HELLO en OLSR) y contiene un algoritmo de carga baja (debido a que es un mapeo simple de velocidad a intervalo de HELLO). El tiempo de procesamiento obtenido en las mediciones fue de 0.191 ms, este valor fue introducido al simulador del caso de estudio 1 a manera de una latencia de procesamiento ocasionada por el *middleware*.

Por último, también se midió el tiempo de inicialización de todo el *middleware* en su conjunto, y en el cual el nodo está inactivo al inicio de sus operaciones en la red. La inicialización del *middleware* comprende desde que éste se ejecuta por el usuario o sistema vehicular (en el prototipo es desde que se inicia el *bundle* del *middleware* en Apache Felix), hasta que el mismo *middleware* inicia las operaciones de los protocolos de red, aplicaciones vehiculares e inicia la ejecución de los modos de adaptación. El tiempo de inicialización del *middleware* en el prototipo fue de 654 ms, éste costo de inicialización de un nodo es significativo y debe ser tomado en consideración por el desarrollador del sistema vehicular.

7.1.3 Costo en memoria y procesamiento de la plataforma de plug-in

Debido a que las plataformas de *plug-ins* encontradas en la comunidad de *software* son sistemas complejos por si mismos, un aspecto importante a considerar es el obtener el impacto y viabilidad de su utilización en un sistema de *software*, en particular un *middleware* dentro de un sistema vehicular. En específico, es de interés saber si utilizar una plataforma de *plug-ins* para comunicar los componentes internos del *middleware* agrega un costo adicional de memoria o procesamiento alto al conjunto de éste.

La implementación de la plataforma de *plug-ins* utilizada en el prototipo de este trabajo, está basada en las tecnológicas OSGi e iPOJO. En este caso particular, además del código escrito por el desarrollador para implementar la interfaz *plug-in* de PLUGAM, el proceso de compilación inserta código OSGi e iPOJO adicional a los archivos .class para dar soporte a la plataforma de *plug-ins*; lo anterior incrementa el tamaño de los archivos binarios de los *plug-ins* (archivo JAR en caso del prototipo) creados por el desarrollador.

Con el propósito de medir el costo de memoria adicional de esta plataforma de *plug-ins*, se realizaron mediciones de la memoria ROM utilizada por cinco *plug-ins* desarrollados para implementar el caso de estudio 1 en el prototipo (véase la figura 28). Cada *plug-in* se compiló y se tomó la medición de la siguiente manera:

- a) Sin agregar la funcionalidad OSGi e iPOJO, es decir haciéndolo un archivo .jar con los archivos con la funcionalidad del *plug-in*.
- b) Solo con la funcionalidad OSGi, es decir haciéndolo un *bundle* OSGi.
- c) Con ambas funcionalidades OSGi e iPOJO, es decir, haciéndolo un servicio iPOJO dentro de un módulo OSGi (llamado *bundle*).

De estas mediciones se encontró que el costo de memoria adicional de OSGi es pequeño, en promedio se agrega 345 bytes o 2.03% al código del *plug-in*; este valor es

		Costo adicional (en %) de memoria ROM, Tamaño archivo jar		Costo adicional (en %) de OSGi/iPOJO sobre archivo .class principal de plug-in
Plug-in	**Sin OSGi/iPOJO**	**Solo con OSGi**	**Con OSGi/iPOJO**	**Tamaño de archive .class**
Tipo contextE (GPS)	31.87 KB	0.952%	18.93%	+80.47%
Tipo algoritmo (velocidad a intervalo HELLO)	4.63 KB	2.88%	52.36%	+63.74%
Tipo algoritmo (vacío, regresa booleano)	3.08 KB	1.075%	69.25%	+89.39%
Tipo protocolo (OLSR)	2759.94 KB	0.039%	0.209%	+73.14%
Tipo aplicación (firstapp)	2.16 KB	5.22%	95.85%	+127.12%
Promedio En KB, %		345 bytes, 2.03%	3.68 KB, 47.31%	8.07 KB, 86.77%

Figura 28. Mediciones sobre el costo de memoria ROM adicional al encapsular a plug-in en bundle OSGi y componente iPOJO

pequeño debido a que crear un *bundle* OSGi solo involucra agregar meta-información de éste en un archivo de texto "de manifiesto" dentro del archivo .jar.

Con respecto al costo memoria adicional causado por iPOJO, se encontró que éste agrega al archivo binario del *plug-in* (el archivo .jar) un promedio de 3.5 KB o 47.31% de memoria ROM con el código específico de iPOJO. Además, se observó que hay una clase java (la que se elige como el componente iPOJO) dentro del archivo .jar del *plug-in* a la cual se le inyecta en tiempo de compilación una mayor cantidad de funcionalidad, instrucciones y métodos de iPOJO. Como se observa en la figura 28, iPOJO incrementa el tamaño de esta clase java principal en 8.07 KB o 86.84%. Cabe resaltar que los archivos .jar incluyen compresión de archivos y así se midió, en cambio las mediciones hechas a los archivos .class no las incluye.

Por otro lado, debido a que la plataforma de *plug-ins* actúa como el canal de comunicación entre los componentes de extensión y el *middleware*, existe también un costo de procesamiento o latencia adicional relacionado a la plataforma *plug-ins*, la cual resulta interesante medir. Para calcular esta latencia se recreó un escenario controlado en el cual se tiene dos componentes iPOJO A y B localizados en diferentes *bundles* o

archivos JAR, en donde A utiliza al componente B. En tiempo de ejecución se mide la latencia desde que el componente A manda llamar a un método de B, hasta que B ejecuta la primera instrucción del método.

Después de realizar 100 mediciones idénticas, la latencia resultante fue de 0.009 ms (9 microsegundos) en promedio, y con desviación estándar de 0.0008 ms. Esta latencia es muy pequeña, casi similar a como si se estuvieran comunicando directamente (0.003 ms); esto quiere decir que el costo de comunicación de la plataforma *plug-ins* OSGi/iPOJO es prácticamente nulo. Sin embargo, se observó que cuando el método se llama por primera vez la latencia resultante es mucho mayor (1.74 ms), esto se debe a que iPOJO inicializa al componente B y lo liga a A hasta el momento que se requiere utilizar al método en tiempo de ejecución (esto se conoce como *"lazy object creation"* en iPOJO), ya las demás veces no existe esta latencia de inicialización del componente.

Actividad	Tiempo de procesamiento (ms)
Creación de objeto no-iPOJO, objeto creador y creado están en mismo bundle	1.91
Creación de objeto iPOJO, objeto creador y creado están en diferentes bundles	2.72
Llamar a método de un objeto no-iPOJO (primera vez)	1.01
Llamar a método de un objeto no-iPOJO (demás veces)	0.003
Llamar a método de un objeto iPOJO (primera vez)	1.74
Llamar a método de un objeto iPOJO (demás veces)	0.009

Figura 29. Mediciones sobre el costo procesamiento adicional utilizando OSGi e iPOJO como base de plataforma de plug-ins

Por último, también se midió el tiempo de inicialización de la plataforma *plug-ins* del prototipo. Específicamente se midió el tiempo que tarda la plataforma desde que el contenedor OSGi Apache Felix es ejecutado por el usuario mediante el comando "java -jar felix.jar", hasta que Apache Felix termina de inicializar, cargar los plug-ins y darle el control al middleware. El tiempo de inicialización resultante fue de 2.4 segundos, de los cuales 0.986 ms son utilizados por Apache Felix para iniciar el contenedor OSGi y

el resto se utiliza para cargar los *plug-ins* depositados en una carpeta específica dentro del código de Apache Felix. Cabe resaltar que el tiempo obtenido es bastante grande, y debe tomarse en consideración como tiempo de inactividad del nodo al iniciar sus operaciones en la red.

En resumen, se encontró que la plataforma de *plug-ins* construida con base en tecnologías OSGi e iPOJO tiene las siguientes características:

- El costo adicional de memoria debido a la encarcelación de los componentes de extension es elevado, éste incrementa el tamaño del código en un 48%, o en 3.5KB. La mayor parte de este incremento de código adicional proviene de iPOJO, OSGi solo aporta unos centenares de bytes.
- En cuanto al costo de procesamiento, en tiempo de ejecución la comunicación entre los componentes utilizando esta plataforma es practicamente nula (10 microsegundos), comparado con otros tiempos de procesamientos de otros aspectos de red en la literatura que andan en el orden de los milisegundos. Sin embargo, la primera vez que los componentes se utilizan los métodos del otro, el costo de procesamiento es de un par de milisegundos. El aspecto negativo de esta plataforma en cuanto a tiempo de procesamiento, es su tiempo de inicialización, la cual es aproximadamente 2 segundos y medio. Ademas, si se agrega el tiempo de inicialización del *middleware* (presentado en la sección 7.1.2), entonces el tiempo final se eleva a 3 segundos; este retraso es crítico y no debe ser ignorado a la hora de emplazar el sistema vehicular en los nodos de la red.

7.1.4 Descripción del costo de procesamiento de extremo a extremo

En el caso de los modos de adaptación *send*, *receive* y *forward*, el *middleware* intercepta el paso del mensaje entre los protocolos de capa de red y aplicaciones dentro del nodo, agregando el costo de procesamiento adicional medido anteriormente. Debido a que el

funcionamiento del *middleware* es distribuido, es decir, existe una instancia del mismo en cada nodo de la red, todavía falta precisar cuál sería el costo de procesamiento o latencia pero de extremo a extremo de la comunicación (conocido en inglés como el *"end-to-end delay"*).

En esta sección presentamos una estimación analítica de la latencia de extremo a extremo de un mensaje que pasa por el *middleware* PLUGAM, utilizando como base las mediciones de costo de procesamiento por nodo descritas anteriormente. La figura 8 de la sección 4.5.1 es útil para visualizar esta latencia extremo a extremo de los modos de adaptación *send*, *receive* y *forward*.

Antes de presentar esta estimación, a continuación se describe de manera informal como ésta es afectada en cada uno de los cuatro modos de adaptación de la arquitectura PLUGAM:

- En el caso de la ejecución de una solución de adaptación en el modo CUPIB, la latencia extremo a extremo no es afectada debido a que esta ejecución es concurrente a la de los protocolos de red, aplicaciones y recorrido de los mensajes entre ellos.
- En todo el recorrido extremo a extremo del mensaje, el modo de adaptación *send* solo agrega un costo de procesamiento en el nodo fuente solamente.
- En el caso del modo *receive*, a través de todo el recorrido extremo a extremo del mensaje, este agrega solamente un costo de procesamiento en el nodo destino.
- En el modo *forward*, la solución de adaptación es ejecutada en cada nodo del recorrido extremo a extremo del mensaje, en consecuencia para obtener la latencia extremo a extremo es necesario multiplicar el costo de procesamiento por el número de retransmisores en el recorrido.

Para realizar una estimación de la latencia extremo a extremo del *middleware*, es necesario conocer el funcionamiento del protocolo de red utilizado y así determinar el

número de retransmisiones por la cuales pasará el mensaje. Es decir, esta estimación depende del funcionamiento del protocolo de red.

A continuación se presenta la estimación analítica del costo adicional en latencia extremo a extremo, pero enfocado a dos tipos de protocolos de red comunes en redes VANET: uno *unicast* y otro de diseminación. Por otro lado, para esta estimación analítica suponemos que solo se definió una configuración de adaptación en alguno de los modos *send*, *receive* y *forward*. Estas estimaciones son mostradas en la tabla 3.

Tabla 3. Estimaciones de la latencia extremo a extremo adicional ocasionada por el middleware PLUGAM, en el caso de un protocolo de red general tipo unicast y otro tipo diseminación de datos.

Modo de adaptación	**Protocolo de red unicast**	**Protocolo de red de diseminación**
Send	$SC * (R) + CS$	$SC * (DR + D + 1) + CS$
Receive	$SC * (R) + CR$	$SC * (DR + D + 1) + CR * D$
Forward	$(SC + CR) * R$	$SC * (DR + D + 1) + CF * DR$

En el caso del protocolo de red *unicast*, se define a R como la cantidad de nodos participando en la ruta del mensaje de extremo a extremo. En el caso del protocolo de diseminación, se define RN como la cantidad de nodos que retransmiten el mensaje en toda la diseminación (este número depende de como funciona el protocolo de diseminación en específico), y se define a DN como número de nodos que reciben el mensaje diseminado y lo transfieren a la aplicación.

Adicionalmente, SC se refiere al costo de procesamiento asociado a una operación de búsqueda dentro del *middleware* requerida para encontrar la configuración de adaptación definida en el modo de adaptación; el costo de procesamiento SC se da inclusive si la búsqueda no tiene éxito debido a que no se definió esa configuración por el usuario. CS simboliza al costo de procesamiento asociado a la ejecución de una solución de adaptación puesta en el modo *send*, CR lo mismo pero puesta en el modo *receive* y CF

en el modo *forward*.

7.2 Simulación de caso de estudio 1

En esta sección se presenta un breve análisis cuantitativo mediante simulación, sobre el impacto de la solución de adaptación propuesta en el caso de estudio 1, construida utilizando la arquitectura PLUGAM. El objetivo es determinar si la solución de adaptación que se propuso, a pesar de ser algo simple, genera un incremento en el desempeño del protocolo de red tal como lo reporta Huang *et al.* (2006).

7.2.1 Descripción de la simulación

En este análisis cuantitativo, esencialmente se compararán a dos protocolos de red, el protocolo OLSR normal y una versión adaptativa del mismo (al cual llamaremos OLSR adaptativo). Esta versión adaptativa incluye la solución de adaptación propuesta para el caso de estudio 1. La simulación se realizó en el simulador de redes Ns-2, ambos protocolos se construyeron utilizando la implementación de OLSR específica para este simulador llamada UM-OLSRv0.8.8. En cuanto a las métricas de la simulación, se compara el desempeño de ambos protocolos obteniendo de la simulación las siguientes métricas:

- El ***"goodput"* promedio** de todas las conexiones de datos entre los nodos participando en la simulación. El *goodput* se define como la cantidad en bytes de datos útiles entregados (a nivel de aplicación), dividido entre el tiempo de transferencia de los datos viajando de extremo a extremo.
- El **costo adicional de enrutamiento normalizado** o en inglés el *"normalized routing overhead"* (NRO), este se define como la relación entre la cantidad de paquetes de control propagados por cada nodo de la red, y la cantidad de paquetes de datos útiles recibido por los nodos. Es decir, esta métrica indica el costo

adicional de un protocolo de red en referencia a la cantidad de paquetes de control que genera el mismo para transportar los datos útiles.

Además de las dos métricas anteriores, también se midió la latencia promedio de extremo a extremo o *"average end-to-end delay"* en inglés. Sin embargo, se decidió omitir esta métrica debido a que los valores obtenidos resultaron muy dispersos o impredecibles entre las corridas de una misma configuración, por consiguiente no resultaban útiles para hacer comparaciones entre los dos protocolos.

El comportamiento anterior se debe a la manera en la que se configuró el escenario de la red VANET simulado. La existencia de semáforos en el mapa complicó la tarea de fijar un valor mayor de velocidad promedio global (VNG), por lo que fue necesario incrementar la aceleración, desaceleración y la velocidad máxima de los vehículos. Elevar estos parámetros de movilidad, en conjunto con los altos totales en semáforos, generan un alto grado de movilidad en el vehículo. Por otro lado, el desempeño de OLSR disminuye considerablemente al tener un grado de movilidad muy alto, lo cual se traduce en una mayor cantidad de paquetes eliminados en nodos intermedios, más aun teniendo un radio de cobertura inalámbrica menor al de omisión (150 m en vez de 250 m). Por lo tanto, los paquetes perdidos se relacionan al cálculo del valor promedio de la latencia extremo a extremo, debido a que los paquetes perdidos no contribuyen al valor promedio de éste y resulta en la obtención de valores de latencia más dispersos o impredecibles.

Cabe resaltar que dadas las métricas de simulación analizadas, se puede considerar que un protocolo de red tiene mejor desempeño que el otro si se cumple lo siguiente:

- Obtener un mayor Goodput, debido a que indica una mayor cantidad de datos útiles.
- Obtener un menor NRO, debido a que indica que se genera un menor tráfico en la red y por ende se consume menos energía en operaciones inalámbricas.

Concerniendo a la implementación del protocolo adaptativo y *middleware* en el simulador, no se implementó el *middleware* PLUGAM completo dentro del simulador, sino que solo se agregó la funcionalidad del modo de adaptación CUPIB dentro del protocolo de red OLSR. Además, se emuló el funcionamiento de este modo tal como lo hace dentro el *middleware* mediante la adición de su costo de procesamiento, traducido como tiempo de espera o latencia de 0.1916 ms (este valor fue obtenido de las mediciones de costo de procesamiento mostradas en la sección 7.1.2).

Por otro lado, el elemento de contexto "velocidad del vehículo" utilizado en la adaptación, es obtenido de la clase MobileNode dentro del ambiente de programación del simulador Ns-2; estos valores de velocidad en la simulación tienen una precisión de 100%, en contraste con los valores que se obtendrían de un GPS de la vida real.

Con el fin de obtener resultados de simulación aproximados a un ambiente VANET, se simuló un escenario de ciudad utilizando datos geográficos reales de las calles de la ciudad de Ensenada, B.C., México, específicamente un área de aproximadamente un kilómetro cuadrado del centro de la ciudad (véase la figura 30). Estos datos geográficos fueron exportados del proyecto Openstreetmap (2005), previo mejoramiento y actualización de los datos por parte de nosotros basándonos en la información de Googlemaps.

Con respecto a la simulación de los vehículos y sus movimientos en el mapa, estos se generaron utilizando la herramienta SUMO (Behrisch *et al.* (2011)). Se eligió SUMO debido a que permitió generar movimientos de vehículos con calles de varios carriles, estos pueden rebasar, frenar si están cerca de otro y simular semáforos en los cruces de las calles. Para esta simulación, se generó en SUMO 41 flujos de vehículos mediante la definición del inicio y fin de la ruta del flujo. SUMO genera automáticamente el movimiento de los vehículos circulando por estos flujos utilizando el algoritmo de ruta más corta de Dijkstra, en donde el peso de las aristas son una combinación de la máxima velocidad de la calle, la distancia al punto final del flujo y la situación del tráfico entre

los vehículos.

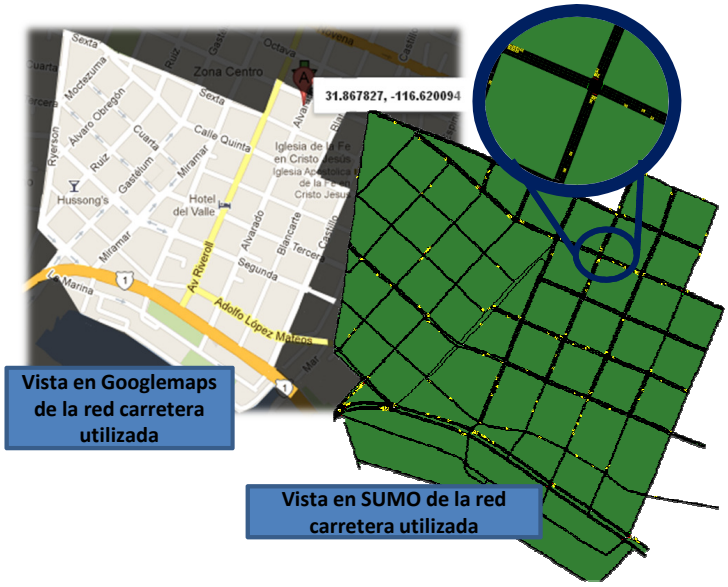

Figura 30. Red carretera utilizada en la simulación del caso de estudio 1

Además, para emular movimientos de vehículos reales, se definieron de los flujos SUMO de tal manera que algunos de estos vehículos entran y salen del área de la simulación. La velocidad máxima permitida en las calles dentro de la ciudad (y por consiguiente de los vehículos) se definió en 40 km/h. El radio de cobertura inalámbrico de los nodos de la red se fijó a 150 metros, en vez del valor por omisión de 250 metros del radio 802.11 de NS-2; se fijó a este valor con el objeto de emular interferencia inalámbrica debido a los edificios de más de un piso que existen en el centro de la ciudad.

Se realizaron varias corridas simulando a los dos protocolos en diferentes escenarios con variaciones en la densidad de red global (DRG) y la velocidad de nodo promedio global (VNG). Por la densidad de red global nos referimos al número total de vehículos en todo el tiempo de la simulación, se utilizaron tres valores de DGR: 100, 150 y 200 vehículos. Solamente se simuló hasta 250 vehículos debido a limitaciones en el equipo de cómputo que corrió las simulaciones y a restricciones de tiempo; por ejemplo, correr una simulación con 250 vehículos tomaba 5 días y con 350 tardaba casi todo un mes.

Extender los resultados de simulación con valores de DGR más altos es deseable y se deja como trabajo a futuro.

El concepto velocidad de nodo promedio global (VNG) se refiere al valor promedio de las velocidades de los vehículos de toda la red y durante todo el tiempo de simulación. Se utilizaron 5 valores de VNG: 10, 15, 20, 25 y 30 km/h. La razón de estos valores de velocidad tan pequeños se debe a que los vehículos están mucho tiempo parados en los semáforos, y estos momentos en 0 km/h reducen el valor promedio al final.

Para representar a los datos de aplicación entre pares de vehículos en la simulación, se utilizaron conexiones de tipo tasa de bits constante, en inglés conocida como "Constant Bit Rate" (CBR). Una conexión CBR consta de un nodo fuente que envía cierta cantidad de datos a una tasa constante a un nodo destino. Las conexiones CBR utilizadas estuvieron activas durante todo el tiempo de simulación. Se decidió generar la máxima cantidad posible de conexiones CBR, la cual sería la mitad del número de total de vehículos en la simulación; sin embargo, esta cifra es menor debido a que en ciertos vehículos entran y salen del área de la simulación, por lo tanto no es posible asociarles una conexión CBR con la duración deseada.

Por último, en la tabla 4 se muestra un resumen de los parámetros de la simulación, incluyendo otros adicionales no mencionados hasta el momento.

7.2.2 Análisis de resultados de la simulación

Los resultados de la simulación, comparando el protocolo OLSR normal contra el OLSR adaptativo, se presentan en las gráficas de la figura 31, éstas incluyen los intervalos de confianza con un nivel de 95%. En las gráficas de la parte superior se mide el *goodput* de los dos protocolos con respecto a los cinco valores de velocidad promedio (VNG), esto

Tabla 4. Parámetros de la simulación del caso de estudio 1

Parámetros de la simulación	
Simulador utilizado	NS-2.34
Implementacion OLSR utilizada	UM-OLSR 0.8.8
Tiempo total de simulación	100 segundos
Área de simulación	Aprox. 1km x 1km
Capa MAC utilizada en los nodos	IEEE 802.11b
Tipo de tráfico generado	CBR/UDP
Tamaño de paquete de datos CBR	512 B
Tasa de datos CBR	100 kps
Cantidad de paquetes por segundo CBR	24
Número de corridas realizadas	5

se realizó para cada uno de los tres escenarios de densidades de red (DGR) tomados a consideración. En las gráficas inferiores se mide el costo adicional de enrutamiento normalizado (NRO) con respecto a los valores de velocidad promedio y en los tres escenarios de densidad de red.

Examinando los resultados mostrados en las gráficas, en primer lugar se observa que los comportamientos de ambos (*goodput* y NRO) son complemente distintos en cada uno de los tres escenarios de densidad de red (DGR). Lo anterior ocurre debido a que fue necesario calibrar la generación de los flujos de vehículos en SUMO de tal manera que diera el valor de densidad de nodos de la red deseada, por lo tanto se generan movimientos de vehículos y rutas de los paquetes diferentes.

Con respecto a la métrica *goodput*, se observa que lo producido por el OLSR adaptativo es muy similar al del protocolo OLSR normal, aunque con valores de *goodput* ligeramente menores en el caso de OLSR adaptativo. Realizando un promedio de todas las mediciones de *goodput* con diferentes valores de DGR y VNG, resulta que en promedio el *goodput* del OLSR adaptativo es 1.42% menor al de OLSR normal; esto significa que la adaptación propuesta en el caso de estudio 1 no genera un mayor desempeño

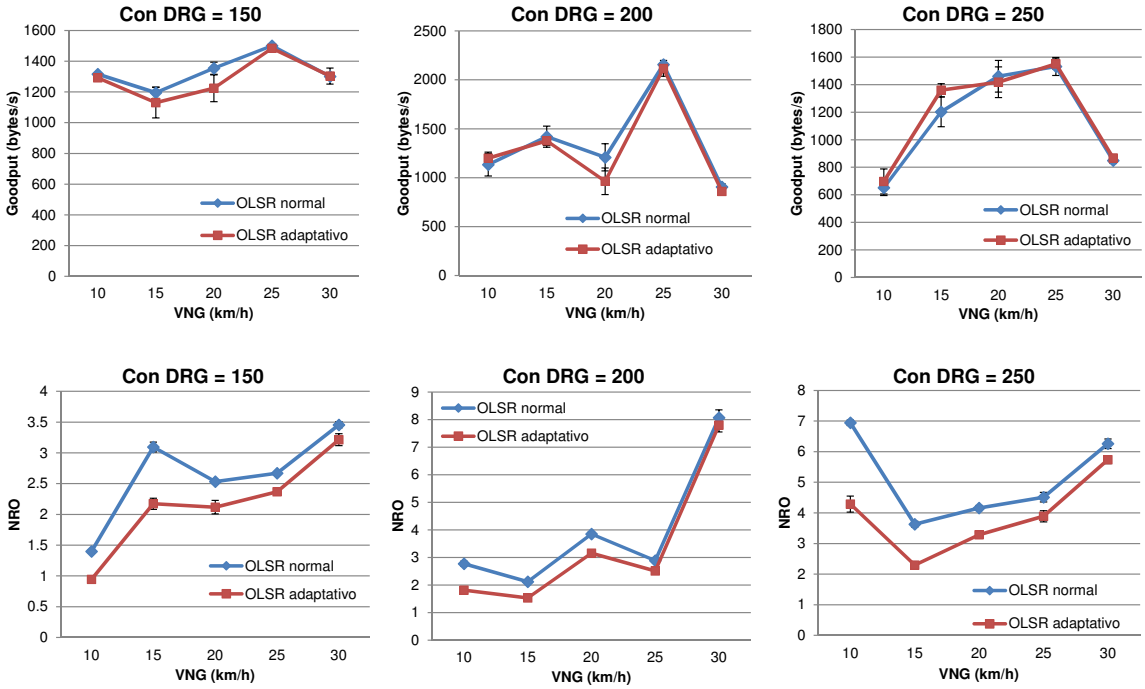

Figura 31. Resultados de la simulación del caso de estudio 1

en *goodput* que la versión por omisión de OLSR, sino que al contrario lo disminuye un poco. Sin embargo, debido a que los intervalos de confianza de los datos no son lo suficientemente pequeños, la anterior conclusión sobre el *goodput* carece de precisión, solo incrementando el número de corridas por punto de simulación (por ejemplo a 10 corridas) se podría aseverar esta conclusión; debido a lo tardado de las simulaciones se dejo esto para trabajo a futuro.

Con respecto a la métrica NRO, aquí se puede observar una clara diferencia entre los dos protocolos examinados. El OLSR adaptativo produce valores de NRO menores a OLSR normal en todos los escenarios de DGR y valores de VNG. Realizando un promedio de todas las mediciones de NRO con los diferentes valores de DGR y VNG, resulta que en promedio el NRO del OLSR adaptativo es 20.78% menor al de OLSR normal; esto significa que la adaptación de caso de estudio 1 genera 20% menos paquetes de control, dado que ambos producen casi el mismo *goodput*.

Los intervalos de confianza en los datos de NRO son lo bastante pequeños para tener certidumbre en la conclusión anterior, las cinco corridas que se realizaron por punto de simulación fueron suficientes. Sin embargo, se puede observar que a mayor valor de velocidad VNG, la diferencia entre los valores de NRO de los dos protocolos OLSR se disminuye.

En resumen, la adaptación del caso de estudio 1 genera una mejora considerable al reducir en 20% la cantidad de datos de control producidos por el protocolo de red OLSR, a un costo de reducir en 2% la cantidad de datos útiles (*goodput*) que puede transportar entre nodos.

7.3 Escalabilidad de la arquitectura y costos adicionales de la solución genérica

Para tener una mayor comprensión del desempeño y la eficiencia de la arquitectura del middleware de adaptación, en esta sección se presenta una descripción sobre su escalabilidad. Además, se hace una comparación entre la arquitectura PLUGAM, la cual es una propuesta genérica, contra un protocolo de adaptación existente, el cual es una solución particular.

7.3.1 Escalabilidad de la arquitectura

Antes de continuar con la discusión sobre escalabilidad, cabe aclarar que cada nodo de la red cuenta con una instancia del *middleware* PLUGAM, ésta es la misma en cada uno de ellos. Es decir, el *middleware* de cada nodo tiene instalado los mismos elementos de contextos, protocolos de red, *plug-ins*, configuraciones de adaptación, aplicaciones, etc.

La escalabilidad de la arquitectura PLUGAM será analizada desde dos aspectos: de acuerdo a la cantidad de nodos que existen en la red y a la cantidad de protocolos de red dentro de cada instancia del *middleware* del nodo.

En referencia a la escalabilidad de la arquitectura PLUGAM con respecto a la cantidad de nodos/vehículos en la red, se observa lo siguiente:

- La arquitectura PLUGAM presentada solo permite crear adaptaciones con efecto local o dentro del nodo, se omite el crear adaptaciones distribuidas. Lo anterior hace a la arquitectura PLUGAM un sistema distribuido descentralizado, lo cual en teoría tiene una mejor escalabilidad que un sistema centralizado. La definición de adaptación local/nodo y distribuida se encuentra en la sección 5.3 de este documento.
- En caso que se quiera extender la arquitectura PLUGAM para lidiar con adaptaciones distribuidas, entonces ésta ya no sería un sistema distribuido descentralizado. Las adaptaciones distribuidas necesitan cierta centralización e interacciones entre nodos para coordinarlas, lo cual seguramente impactaría en la escalabilidad de la arquitectura con respecto a la cantidad de nodos.

Referente a la escalabilidad de la arquitectura PLUGAM, pero ahora con respecto a la cantidad de protocolos de red utilizados en el *middleware*, el aspecto genérico de ésta produce un costo adicional en contraste a una propuesta de un protocolo adaptativo en particular. Por cada protocolo de red agregado y utilizado en el *middleware*, se generan los siguientes costos adicionales debido a que es un componente de extensión:

- Se genera un costo en memoria adicional debido a la necesidad de incrustar meta-información sobre el componente de extensión, la cual permite al *middleware* utilizarlo y manejarlo. Este costo está relacionado al diseño modular y bajo acoplamiento de los componentes de extensión dentro de la arquitectura PLUGAM.

- Un protocolo de red, así como cualquier otro componente de extensión, es encapsulado en algún *plug-in*. La construcción de un plug-in necesita de instrucciones y código adicional que permite a éste comunicarse con la plataforma de *plug-ins* y el *middleware*. Además, la implementación de los métodos de la interfaz *plug-in* tipo protocolo requiere código adicional que se traduce en un costo en memoria adicional. Las mediciones realizadas en la sección 7.1.3 de este documento indican que este costo adicional debido a la encapsulación del protocolo de red en un *plug-in*, incrementa 3.5 KB el tamaño del código.

En relación a tener un gran número de soluciones de adaptación definidas en el *middleware*, esto eleva la probabilidad de que existan interferencias entre los efectos de estas adaptaciones, lo cual produciría resultados inesperados. Por lo tanto, es recomendable ser cuidadoso y experimentar bastante con las soluciones de adaptación hasta lograr calibrar sus efectos y tener los resultados esperados.

En el caso de la plataforma de *plug-in* elegida para el prototipo de la arquitectura PLUGAM, la cual está basada en tecnología OSGi e iPOJO, no se encontraron en la literatura análisis rigurosos sobre su escalabilidad, sin embargo OSGi es utilizado en la construcción de programas con una gran cantidad de módulos OSGi (por ejemplo el ambiente de programación Eclipse) y mantiene un buen desempeño.

7.3.2 Costos adicionales debido a la generalidad de la solución

Con motivo de tener una idea general sobre el desempeño y eficiencia de la arquitectura PLUGAM, en esta sección se habla sobre el costo adicional generado por un protocolo adaptativo construido por la arquitectura, la cual es una solución genérica, en contraste con construirlo sin ayuda de la arquitectura. Se utilizará como ejemplo la propuesta de protocolo adaptativo de Chen *et al.* (2012), la cual realiza una adaptación basada en hacer un cambio entre dos protocolos de red diferentes; para mayor información sobre este protocolo véase la tabla 1 de la sección 3.1 de este documento. Este costo adicional

generado por ser una solución genérica es la siguiente:

- Existe un costo adicional de procesamiento debido al acceso indirecto entre los componentes de extensión involucrados en una solución de adaptación, contrario a un protocolo adaptativo particular como el de Chen *et al.* (2012) donde estos elementos están integrados y con comunicación directa entre ellos. Este costo de procesamiento depende en gran medida de la latencia introducida por la comunicación entre *plug-ins* y el *middleware*, pasando por la plataforma de *plug-ins*; en el prototipo implementado en este trabajo, las mediciones de la sección 7.1.3 muestran que esta latencia resultó muy pequeña, aproximadamente 0.009 ms.
- Debido a que el *middleware* agrega un encabezado de 132 bytes a los mensajes de las aplicaciones, este costo adicional de memoria debe tomarse en cuenta ya que reduce la cantidad de información útil que pueden transportar los protocolos de red y aplicaciones vehiculares. Este encabezado es utilizado por el *middleware* para identificar el protocolo de red y la aplicación asociada al mensaje, en un entorno de múltiples aplicaciones y protocolos de red propuesto en la arquitectura PLUGAM. La propuesta adaptativa no genérica de Chen *et al.* (2012) integra los dos protocolos de red en un solo super protocolo, por lo tanto no necesita la información del encabezado del *middleware*.
- Existe un costo de procesamiento adicional debido a la puesta en marcha y ejecución de una solución de adaptación. En un protocolo adaptativo no genérico como el de Chen *et al.* (2012) tienen acceso directo a los valores de los elementos de contexto, solo necesitan ejecutar el algoritmo de adaptación. Con base en las mediciones de la sección 7.1.2, se infiere que este tiempo de procesamiento adicional es una cifra que tiene como cota superior 0.2 ms.
- Por último, protocolos adaptativos no genéricos como el propuesto por Chen *et al.* (2012) no cuentan con el subsistema manejador de contexto, el cual introduce un costo de procesamiento adicional concurrente a la ejecución de las soluciones de adaptación, protocolos de red y aplicaciones.

7.4 Resumen del capítulo

En este capítulo se presentó la evaluación del prototipo del *middleware* de adaptación.

Primeramente se realizó un análisis de costo/desempeño al prototipo, en el cual se midieron diferentes parámetros, entre estos los requerimientos de memoria RAM y ROM del *middleware*. La medición de este parámetro arrojó que los requerimientos del *middleware* son moderados, incluso compatibles con los de las aplicaciones para dispositivos móviles con sistema operativo Android (2007).

Enseguida se calculó el costo del procesamiento (medido en tiempo) generado por el *middleware* al ejecutar una solución de adaptación en los diferentes modos de adaptación, como resultado se obtuvo que el tiempo de procesamiento de los cuatro modos de adaptación va desde una décima de milisegundo hasta 1.6 ms como máximo, el tiempo de procesamiento de la solución de adaptación medida fue de 0.191ms.

Adicionalmente se midió el costo en memoria ROM y de procesamiento de la plataforma de *plug-ins* que brinda los componentes de extensión al *middleware*, se obtuvo que el tiempo de procesamiento que genera la plataforma para comunicar el *middleware* con los componentes de extensión es prácticamente nula aproximadamente de 10 microsegundos, y el de inicialización de la plataforma en el prototipo fue de 2.4 segundos lo que se considera alto, y un punto negativo de ésta.

Por otra parte se presentó la simulación realizada al caso de estudio 1 que muestra como se agrega la funcionalidad de adaptación a un protocolo de capa de red predefinido, el objetivo de ésta fue determinar si la solución de adaptación incrementa el desempeño del protocolo, para hacer esto se realizó un análisis cuantitativo en el cual se compara un protocolo de red contra la versión adaptativa del mismo. Se obtuvo como resultado que la solución de adaptación implantada en el protocolo de red reduce

en un 20% la cantidad de datos de control en comparación con la versión por omisión del protocolo, y disminuye en un 2% la cantidad de datos útiles que puede transportar entre nodos y entregar a la aplicación.

Finalmente, se determinó la escalabilidad de la arquitectura PLUGAM de acuerdo a la cantidad de nodos que existen en la red y a la cantidad de protocolos de red dentro de cada instancia del *middleware* del nodo.

En el siguiente capítulo se presentan las conclusiones, limitaciones, aportaciones y trabajo futuro de este trabajo de tesis.

Capítulo 8

Conclusiones y trabajo a futuro

Este capítulo concluye la investigación realizada y resume las contribuciones más importantes de la misma. Adicionalmente, se presentan las líneas de investigación relacionadas a la presente tesis que pueden ser exploradas a futuro.

Las redes VANET representan un gran reto para los protocolos de capa de red debido a sus características dinámicas y a las diferentes necesidades de las aplicaciones vehiculares. Debido a lo anterior, en este trabajo se propone el diseño de una arquitectura *middleware* (PLUGAM) que permite adaptar el comportamiento de la capa de red del sistema vehicular, utilizando información del contexto local. Además, debido a que las necesidades de las aplicaciones vehiculares no pueden ser atendidas por un solo tipo de protocolo de capa de red, esta arquitectura promueve la co-existencia de múltiples protocolos de red en el sistema vehicular.

La arquitectura PLUGAM tiene como núcleo un modelo de adaptación propuesto en este trabajo. Aunque la arquitectura se centra en explorar dos tipos de adaptaciones clave (adaptación del comportamiento interno de protocolo de red en base a contexto, y adaptaciones en base a contexto involucrando múltiples protocolos), el modelo permite extender la arquitectura para lidiar con más tipos de adaptación en la capa de red.

La arquitectura PLUGAM es genérica, es decir está diseñada para crear una capa de red adaptativa con elementos de contexto, protocolos de red, algoritmos de adaptación y aplicaciones vehiculares definidos por el usuario. Para lograr lo anterior, el *middleware* PLUGAM debe ser instanciado al cargarle los elementos necesarios de la capa de red adaptativa (llamados componentes extensión), estos componentes son introducidos al *middleware* utilizando la tecnología plug-ins.

8.1 Conclusiones

Las conclusiones de este trabajo se describen a continuación, estas están organizadas de acuerdo a los objetivos específicos (A1 a A5) planteados en la sección 1.2.2 de este trabajo de tesis.

A1: El diseño de la arquitectura de adaptación se apoyó en el uso conceptos de ingeniería de *software* como componentes de *software*, separación de preocupaciones y tecnologías *plug-ins*, lo que simplifica la tarea de diseñar y construir una capa de red adaptativa haciendo uso de la arquitectura PLUGAM propuesta en este trabajo. Por ejemplo, el uso de componentes de software ha sido citado en trabajos como Ramdhany *et al.* (2009), Chefrour (2005), Hajj *et al.* (2010) y en Pour (1998). El concepto de separación de preocupaciones ha sido utilizado en trabajos como Marvie *et al.* (2002), Montanari *et al.* (2003), Alford (1994) y en Herrmann y Mezini (2000). Por último, la tecnología *plug-ins* ha sido aplicada en trabajos como Lei *et al.* (2011), Sunderam y Kurzyniec (2002), Sun *et al.* (2010) y en Wang *et al.* (2011).

A2: La arquitectura PLUGAM se diseñó principalmente para que pudiera ofrecer dos tipos de adaptaciones que se consideraron relevantes. Para esto se diseñaron los cuatro modos de adaptación y acciones de adaptación especiales que permitieron realizar las adaptaciones consideradas, y además generar varios de los protocolos adaptativos de la literatura. Esta arquitectura es fácilmente extendible debido a que permite agregar nuevos modos y acciones de adaptación especiales (tal como aquellos propuestos en el capítulo 5), estas extensiones lograrían aumentar considerablemente el conjunto de protocolos adaptativos que se pueden crear con la arquitectura.

A3: Con el propósito de incorporar a la arquitectura la funcionalidad para que los

protocolos de capa de red modifiquen su comportamiento con base en la información de contexto, fue propuesto el modo de adaptación CUPIB. El funcionamiento de CUPIB es un simple mecanismo de encuesta (*polling*), el cual espera cierto intervalo de tiempo, ejecuta el algoritmo correspondiente e informa los valores de salida asociados a parámetros de un protocolo de red, con lo cual se logra esta funcionalidad. El mecanismo descrito anteriormente es muy simple, pero agregándole más funcionalidad puede llegar a resolver casos más complejos, por ejemplo que ejecute el algoritmo de adaptación cuando cambien algún valor de entrada.

A4: La arquitectura contempla una entidad llamada elementos de contexto, cada uno es un solo dato de información contextual, con su tipo de dato asociado. También se definieron las fuentes de contexto que proveerán al *middleware* de estos elementos mediante dos mecanismos: encuesta (*polling*) y publicación y suscripción. Además la arquitectura define un módulo dentro del *middleware* llamado manejador de contextos, el cual se encarga de obtener los valores actuales de los elementos de contexto al interactuar con las fuentes de estos datos. Este entorno de contextos puede ser extendido agregando conceptos utilizados en otras propuestas de arquitecturas, que se especializan en proveer información de contexto a aplicaciones en ambientes móviles.

A5: El objetivo de la evaluación de este trabajo de tesis se centró en valorar la viabilidad de la arquitectura del PLUGAM. De los resultados de las diferentes actividades de evaluación se concluye lo siguiente:

- Los requerimientos de memoria de *middleware* de adaptación (al menos del prototipo) son moderados, incluso se podría implementar en dispositivos móviles de media gama con sistema operativo Android (2007).

- El tiempo de procesamiento introducido por el *middleware* para ejecutar una solución de adaptación, en cualquiera de los modos de adaptación, se encuentra en el intervalo de 0.5 a 2 milisegundos. Este tiempo de procesamiento del nodo

es razonable en comparación con otros trabajos de protocolos de comunicación, ya que también se encuentran en el orden de los milisegundos.

- Debido a que la arquitectura PLUGAM es un sistema distribuido completamente descentralizado, es escalable en cuanto al número de nodos en la red. En referencia a la cantidad de protocolos de red coexistiendo en el sistema vehicular, su escalabilidad se ve afectada por el costo en memoria generado por cada protocolo.

- Con base en las mediciones hechas a la plataforma de *plug-ins* del prototipo, se concluye que los costos de procesamiento en tiempo de ejecución son prácticamente nulos (10 microsegundos), lo cual es una característica buena de la plataforma. Sin embargo, los costos se trasladan al tiempo de inicialización de la plataforma y de memoria generados por la misma. Aun así, las tecnologías OSGi/iPOJO resultan una opción buena como plataforma de *plug-ins* de un sistema de comunicación vehicular.

- El análisis realizado a la información que arroja la simulación de la solución de adaptación del caso de estudio 1, indica que si se utiliza en la arquitectura una solución de adaptación previamente definida y probada, como la que fue seleccionada de la literatura para realizar la simulación, la arquitectura arrojará buenos resultados a pesar de sus costos internos adicionales. Los resultados de la simulación indican una mejora en desempeño en cuanto a la cantidad de datos de control generados (se disminuyen en 20%), en compensación con disminuir en un 2% la cantidad de datos útiles que se pueden transportar en la red.

En perspectiva, la arquitectura propuesta incluye un entorno donde coexisten en la capa de red múltiples protocolos en paralelo, además de poder realizar adaptaciones que cambien o seleccionen entre el uso de uno de ellos en tiempo de ejecución. Sin embargo, debido a este funcionamiento solo fue probado en condiciones limitadas y en un entorno de pila de protocolos y varios tipos de protocolos de red reales, se decidió no poner este punto como un objetivo específico de esta tesis. Aun así, vale la pena

describir cómo la arquitectura PLUGAM logra este funcionamiento.

La definición del entorno de múltiples protocolos de red en la arquitectura propuesta involucró lo siguiente: creación de un encabezado del mensaje con información útil para el *middleware*, intercepción de mensajes entre aplicaciones y protocolos de red, e interacción entre las aplicaciones y el *middleware* mediante la creación de dos interfaces (IAplication e IMiddlewareProxy). Además, con el fin de simplificar el entorno los protocolos de capa de red se ejecutan de manera aislada, no existe interacción entre ellos. Debido a lo anterior, posibles interferencias del canal en comunicación por los protocolos deben ser solucionadas fuera del *middleware* por el desarrollador, éste tema es complejo y sería un interesante trabajo a futuro. La selección entre protocolos de capa de red se realiza en ciertos momentos del trayecto extremo a extremo de mensaje, para ésto se definieron en la arquitectura los modos *send*, *receive*, *forward*, además de las acciones de adaptación especiales *filter* y *switcher*.

8.2 Contribuciones

A continuación se menciona las aportaciones principales de este trabajo de investigación:

- Un modelo de adaptación genérico para protocolos de capa de red, el cual propone separar la lógica adaptativa de la funcionalidad específica del protocolo de red, y la divide en diferentes elementos que conforman la adaptación, entre estos la información obtenida del contexto local del nodo.
- Se propuso un entorno para el manejo de múltiples protocolos en la capa de red de un sistema vehicular en VANETs, y se implementaron soluciones de adaptación utilizando a este conjunto de protocolos.
- El diseño de una arquitectura distribuida basada en el modelo adaptativo, el cual disminuye la complejidad de construir una capa de red adaptativa con base en

conceptos de ingeniería de software como componentes de *software*, separación de preocupaciones, tecnología *plug-ins*.

- Se utilizó la tecnología *plug-ins* para la construcción de un sistema vehicular y su capa de red. Tecnología cuyo uso no había sido explorado para este tipo de sistemas y que mostró ser una buena opción para agregar módulos independientes al mismo, que pueden ser reutilizables en diferentes implementaciones.
- El desarrollo de un prototipo de la arquitectura PLUGAM, el cual puede ser utilizado como sistemas de pruebas por otros investigadores para proponer y probar sus soluciones adaptativas en protocolos de capa de red.

Derivado de este trabajo de tesis se publicó un artículo de revista que a continuación se cita:

Nombre de la publicación: *PLUGAM: Plug-in based Adaptation Middleware for Network Layer Protocols for Vehicular Ad Hoc Networks*

Nombre de la revista: *International Journal of Distributed Sensor Networks Volume 2013, reconocida en el Science Citation Index y Journal Citation Report*

Editorial: *Hindawi*

Fecha de aceptación: *4 de enero de 2013*

Fecha de publicación: *24 de mayo de 2013*

8.3 Limitaciones del presente trabajo

- La arquitectura *middleware* solo permite realizar adaptaciones locales al nodo, la intención es que los cambios en los nodos produzcan la adaptación de forma global. Existen algunas propuestas adaptativas complejas que involucran el uso de adaptaciones distribuidas, las cuales no pueden ser recreadas en la arquitectura PLUGAM.

- Existe la posibilidad de que haya interferencia entre dos o más protocolos de red que utilizan un mismo canal de comunicación, esta situación no la pueden resolver el *middleware* PLUGAM. Éste supone que el desarrollador de los protocolos de capa de red tiene identificado qué protocolos serán cargados en el sistema, de tal manera que elimine cualquier posibilidad de interferencia.
- El entorno de contextos no toma en cuenta casos como obtener valores anteriores de los elementos de contextos (por ejemplo, obtener el promedio de los últimos 10 valores). Además no incorpora un mecanismo para fusión de contextos, ni una ontología de contextos que permita intercambiar contextos similares o tener diferentes implementaciones de un mismo tipo.
- El impacto de una solución de adaptación implementada por la arquitectura debe ser validado manualmente por el desarrollador. La arquitectura PLUGAM no ofrece mecanismos que permitan verificar que la solución de adaptación creada no produzca resultados malos o inesperados.

8.4 Trabajo a futuro

Se propone como trabajo a futuro lo siguiente:

- Extender la arquitectura para poder realizar adaptaciones distribuidas.
- Extender el entorno de contexto introduciendo elementos de contexto asociados al mensaje, ontología de contextos, y fusionar otros contextos de alto nivel.
- Explorar mecanismos que permitan validar o informar sobre los efectos de las soluciones de adaptación en el sistema vehicular en tiempo de ejecución. Esto puede ayudar a acelerar la calibración de las soluciones de adaptación construidas con la arquitectura *middleware*, de tal manera que tengan un efecto positivo y mejoren el desempeño.
- Introducir un mecanismo para lidiar con posibles interferencias entre protocolos de red.

- Extender la arquitectura proponiendo nuevos modos de adaptación, en específico los descritos en la sección 5.2.3 de este documento.
- Mejorar el aspecto gráfico del *middleware* de adaptación, de tal manera que se puedan ver los componentes de extensión disponibles y armar las configuraciones de adaptación visualmente. Con base en ésto realizar una evaluación cualitativa y medir la usabilidad.
- Agregar funcionalidad para que en tiempo de ejecución las aplicaciones vehiculares puedan acceder directamente a los elementos de contextos. Con ésto las aplicaciones podrán implementar sus propias adaptaciones si lo requieren, haciendo al *middleware* de adaptación más flexible.

Referencias bibliográficas

Abdalla, G., AbuRgheff, M., y Senouci, S. (2007). Current Trends in Vehicular Ad Hoc Networks. En *International Workshop on ITS for Ubiquitous Roads (UBIROADS)*. IEEE GIIS.

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M., y Steggles, P. (1999). Towards a Better Understanding of Context and Context-Awareness. En *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, HUC '99, p. 304–307, London, UK, UK. Springer-Verlag. ISBN 3-540-66550-1.

Abuelela, M. y Olariu, S. (2007). Traffic-adaptive packet relaying in VANET. En *Proceedings of the fourth ACM international workshop on Vehicular ad hoc networks*, VANET '07, p. 77–78, New York, NY, USA. ACM. ISBN 978-1-59593-739-1.

Adler, C., Eichler, S., Kosch, T., Schroth, C., y Strassberger, M. (2006). Self-organized and Context-Adaptive Information Diffusion in Vehicular Ad Hoc Networks. En *3rd International Symposium on Wireless Communication Systems ISWCS '06*, p. 307–311.

Al-Doori, M. M., Al-Bayatti, A. H., y Zedan, H. (2010). Context aware architecture for sending adaptive HELLO messages in VANET. En *Proceedings of the 4th ACM International Workshop on Context-Awareness for Self-Managing Systems*, Vol. 10 de *CASEMANS '10*, p. 65–68, New York, NY, USA. ACM. ISBN 978-1-4503-0213-5.

Alford, M. (1994). Attacking requirements complexity using a separation of concerns. En *Requirements Engineering, 1994., Proceedings of the First International Conference on*, p. 2–5.

Android (2007). Página web de sistema operativo Android de Google. Recuperado 22 de octubre de: http://www.android.com/.

Apache Felix (2006). Página web de Apache Felix. Recuperado 22 de octubre de: http://felix.apache.org/site/index.html.

Apache ServiceMix (2008). Página web de Apache ServiceMix. Recuperado 22 de octubre de: http://servicemix.apache.org/home.html.

Bai, F., Sadagopan, N., y Helmy, A. (2003). The IMPORTANT framework for analyzing the Impact of Mobility on Performance Of RouTing protocols for Adhoc NeTworks. *Ad Hoc Networks*, **1**(4): 383–403.

Behrisch, M., Bieker, L., Erdmann, J., y Krajzewicz, D. (2011). SUMO - Simulation of Urban MObility: An Overview. En *SIMUL 2011, The Third International Conference on Advances in System Simulation*, Barcelona, Spain.

Benslimane, A. (2004). Optimized Dissemination of Alarm Messages in Vehicular Ad-Hoc Networks (VANET). En Z. Mammeri y P. Lorenz, editores, *High Speed Networks and Multimedia Communications*, Vol. 3079 de *Lecture Notes in Computer Science*, p. 655–666. Springer Berlin Heidelberg. ISBN 978-3-540-22262-0.

Boleng, J., Navidi, W., y TracyCamp (2002). Metrics to Enable Adaptive Protocols for Mobile Ad Hoc Networks. En *Proceedings of the International Conference on Wireless Networks ICWN '02*, p. 293–298.

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

Car 2 Car Communication Consortium (2007). Página web del Car 2 Car Communication Consortium. Recuperado 19 de Marzo de: http://www.car-2-car.org/.

CarTALK2000 (2001). Página web de proyecto CarTALK 2000. Recuperado 19 de Marzo de: http://www.cartalk2000.net/.

Chefrour, D. (2005). Developing component based adaptive applications in mobile environments. En *Proceedings of the 2005 ACM symposium on Applied computing*, SAC '05, p. 1146–1150, New York, NY, USA. ACM. ISBN 1-58113-964-0.

Chen, Y.-S., Hsu, C.-S., y Jiang, Y.-T. (2012). A delay-bounded routing protocol for vehicular ad hoc networks with traffic lights. En *Communication, Networks and Satellite (ComNetSat), 2012 IEEE International Conference on*, p. 122–126.

Cheng, A. y Rajan, K. (2003). A digital map/GPS based routing and addressing scheme for wireless ad-hoc networks. En *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE*, p. 17–20.

Cheng, P.-C., Weng, J.-T., Tung, L.-C., Lee, K. C., Gerla, M., y Hårri, J. (2010). GeoDTN+Nav: A Hybrid Geographic and DTN Routing with Navigation Assistance in Urban Vehicular Networks. En *1st International ICST Symposium on Vehicular Computing Systems (ISVCS)*, p. 1–10.

Chennikara-Varghese, J., Chen, W., Altintas, O., y Cai, S. (2006). Survey of Routing Protocols for Inter-Vehicle Communications. En *Mobile and Ubiquitous Systems: Networking Services, 2006 Third Annual International Conference on*, p. 1–5.

Clausen, T. y Jacquet, P. (2003). RFC 3626: Optimized Link State Routing Protocol. Recuperado 22 de octubre de: http://www.ietf.org/rfc/rfc3626.txt.

ComputerWorld (2013). Página web donde se habla de la tendencia de futuros sistemas vehiculares a ser open source. Recuperado 26 de septiembre de: http://www.computerworld.com/s/article/9243075/Your_car_is_about_to_go_open_source.

COOPERS (2006). Página web de proyecto COOPERS. Recuperado 21 de marzo de: http://www.coopers-ip.eu/index.php?id=project.

Crockford, D. (2006). RFC 4627: The application/json Media Type for JavaScript Object Notation (JSON). Recuperado 22 de octubre de: http://tools.ietf.org/html/rfc4627.

CVIS (2006). Página web de proyecto CVIS. Recuperado 21 de marzo de: http://www.cvisproject.org/.

Day, J. y Zimmermann, H. (1983). The OSI reference model. *Proceedings of the IEEE*, **71**(12): 1334–1340.

Delot, T., Cenerario, N., y Ilarri, S. (2010). Vehicular event sharing with a mobile peer-to-peer architecture. *Transportation Research Part C: Emerging Technologies*, **18**(4): 584–598.

Dey, A. K. (2001). Understanding and Using Context. *Personal Ubiquitous Comput.*, **5**(1): 4–7.

Eclipse (2004). Página web de Eclipse. Recuperado 29 de mayo de: http://www.eclipse.org/.

Eichler, S., Schroth, C., Kosch, T., y Strassberger, M. (2006). Strategies for context-adaptive message dissemination in vehicular ad hoc networks. En *Proceedings of the Second International Workshop on Vehicle-toVehicle Communications*, p. 1–9.

Enkelmann, W. (2003). FleetNet - applications for inter-vehicle communication. En *Proceedings of the 2003 IEEE Intelligent Vehicles Symposium*, p. 162–167.

Entune (2011). Página web de sistema Entune de Toyota. Recuperado 21 de marzo de: http://www.toyota.com/entune/.

Equinox (2005). Página web del contenedor OSGi Equinox. Recuperado 22 de octubre de: http://www.eclipse.org/equinox/.

Extensiones de Chrome (2012). Página web de descripción general de la plataforma de extensiones de Google Chrome. Recuperado 22 de octubre de: http://developer.chrome.com/extensions/overview.html.

Festag, A., Noecker, G., Strassberger, M., Lübke, A., y Bochow, B. (2008). NoW – Network on Wheels: Project Objectives, Technology and Achievements. En *Proceedings of the 5th International Workshop on Intelligent Transportation (WIT)*, p. 211–216.

Fübler, H., Widmer, J., Käsemann, M., Mauve, M., y Hartenstein, H. (2003). Contention-based forwarding for mobile ad hoc networks. *Ad Hoc Networks*, **1**(4): 351 – 369.

Garber, L. (2002). Will 3G really be the next big wireless technology. *Computer*, **35**(1): 26–32.

Genivi (2009). Página web donde se indica los productos compatibles Genivi para la construcción de futuros sistemas vehiculares. Recuperado 26 de septiembre de: http://www.genivi.org/compliant-products.

Giannoulis, S., Katsanos, C., Koubias, S., y Papadopoulos, G. (2004). A hybrid adaptive routing protocol for ad hoc wireless networks. En *Proceedings of IEEE International Workshop on Factory Communication Systems WFCS'04*, p. 287–290.

GPSylon (2009). Página web de projecto GPSylon. Recuperado 22 de octubre de: http://www.tegmento.org/gpsylon/, http://www.tegmento.org/gpsylon/.

Grace, P. (2009). Dynamic Adaptation. En H. M. B. Garbinato y L. Rodrigues, editores, *Middleware for Network Eccentric and Mobile Applications*, p. 285–304. Springer.

Grace, P., Coulson, G., Blair, G., Porter, B., y Hughes, D. (2006). Dynamic reconfiguration in sensor middleware. En *Proceedings of the international workshop on Middleware for sensor networks*, MidSens '06, p. 1–6, New York, NY, USA. ACM. ISBN 1-59593-424-3.

Hadzic, I., Marcus, W. S., y Smith, J. M. (1999). Policy and Mechanism in Adaptive Protocols. Reporte técnico MS-CIS-01-03, Department of Computer and Information Science University of Pennsylvannia and Bellcore.

Hajj, H. M., El-Hajj, W., El Dana, M., Dakroub, M., y Fawaz, F. (2010). An extensible software framework for building vehicle to vehicle applications. En *Proceedings of the 6th International Wireless Communications and Mobile Computing Conference*, IWCMC '10, p. 26–31, New York, NY, USA. ACM. ISBN 978-1-4503-0062-9.

Herrmann, S. y Mezini, M. (2000). PIROL: a case study for multidimensional separation of concerns in software engineering environments. *SIGPLAN Not.*, **35**(10): 188–207.

Huang, Y., Bhatti, S. N., y Parker, D. (2006). Tuning OLSR. En *Proceedings of the IEEE 17th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, p. 1–5. IEEE.

Ibrahim, K., Weigle, M., y Abuelela, M. (2009). p-IVG: Probabilistic Inter-Vehicle Geocast for Dense Vehicular Networks. En *Vehicular Technology Conference, 2009. VTC Spring 2009. IEEE 69th*, p. 1–5.

IEEE 802.11p (2010). Página web del estandar IEEE 802.11p (versión 2010). Recuperado 20 de marzo de: http://standards.ieee.org/findstds/standard/802.11p-2010.html.

Intelligent Transportation System (2005). Página web de aplicaciones de ITS. Recuperado 19 de Marzo de: http://www.itsoverview.its.dot.gov/.

IntelNews (2009). Página web donde se habla de la participacion de Intel en la construccion de futuros sistemas vehiculares. Recuperado 26 de septiembre de: http://newsroom.intel.com/community/intel_newsroom/blog/2011/11/09/intel-toyota-drive-joint-research-on-next-generation-in-vehicle-infotainment-systems.

iPOJO (2008). Página web de iPOJO. Recuperado 22 de octubre de: http://felix.apache.org/site/apache-felix-ipojo.html.

Jacquet, P., Minet, P., Laouiti, A., Viennot, L., Clausen, T., y Adjih, C. (2001). IETF Internet Draft: Multicast Optimized Link State Routing (MOLSR). Recuperado 22 de octubre de: http://tools.ietf.org/html/draft-jacquet-olsr-molsr-00.

Jiang, D., Taliwal, V., Meier, A., Holfelder, W., y Herrtwich, R. (2006). Design of 5.9 ghz dsrc-based vehicular safety communication. *Wireless Communications, IEEE*, **13**(5): 36–43.

Johnson, D., Hu, Y., y Maltz, D. (2007). RFC 4728: The Dynamic Source Routing Protocol. Recuperado 22 de octubre de: http://www.ietf.org/rfc/rfc4728.txt.

Karp, B. y Kung, H. T. (2000). GPSR: greedy perimeter stateless routing for wireless networks. En *Proceedings of the 6th annual international conference on Mobile computing and networking*, MobiCom '00, p. 243–254, New York, NY, USA. ACM. ISBN 1-58113-197-6.

Kim, G., Shin, D., y Shin, D. (2004). Design of a Middleware and HIML (Human Interaction Markup Language) for Context Aware Services in a Ubiquitous Computing Environment. En L. Yang, M. Guo, G. Gao, y N. Jha, editores, *Embedded and Ubiquitous Computing*, Vol. 3207 de *Lecture Notes in Computer Science*, p. 682–691. Springer Berlin Heidelberg. ISBN 978-3-540-22906-3.

Knopflerfish (2005). Página web del contenedor OSGi Knopflerfish. Recuperado 22 de octubre de: http://www.knopflerfish.org/.

Ko, Y.-B. y Vaidya, N. H. (2000). Location-aided routing (LAR) in mobile ad hoc networks. *Wireless Networks*, **6**(4): 307–321.

Korkmaz, G., Ekici, E., Özgüner, F., y Özgüner, U. (2004). Urban multi-hop broadcast protocol for inter-vehicle communication systems. En *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*, VANET '04, p. 76–85, New York, NY, USA. ACM. ISBN 1-58113-922-5.

Kosch, T. (2005). Phase-Transition Phenomena with Respect to the Penetration Rate of DSRC Enabled Vehicles. En *12th World Congress on Intelligent Transportation Systems (ITS)*.

Kurose, J. F. y Ross, K. W. (2009). *Computer Networking: A Top-Down Approach*. Addison-Wesley Publishing Company, USA, quinto edición. ISBN 0136079679, 9780136079675.

LeBrun, J., Chuah, C.-N., Ghosal, D., y Zhang, M. (2005). Knowledge-based opportunistic forwarding in vehicular wireless ad hoc networks. En *Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st*, Vol. 4, p. 2289–2293.

Lee, K., Haerri, J., Lee, U., y Gerla, M. (2007). Enhanced Perimeter Routing for Geographic Forwarding Protocols in Urban Vehicular Scenarios. En *Globecom Workshops, 2007 IEEE*, p. 1–10.

Lee, K., Lee, U., y Gerla, M. (2010). Geo-opportunistic routing for vehicular networks [Topics in Automotive Networking]. *Communications Magazine, IEEE*, **48**(5): 164–170.

Lei, G., Yu, F., y Shulin, P. (2011). Software Framework Construction Based on Plug-in Technology. En *Computational and Information Sciences (ICCIS), 2011 International Conference on*, p. 762–764.

Leinmüller, T., Maihöfer, C., Schoch, E., y Kargl, F. (2006). Improved security in geographic ad hoc routing through autonomous position verification. En *Proceedings of the 3rd international workshop on Vehicular ad hoc networks*, VANET '06, p. 57–66, New York, NY, USA. ACM. ISBN 1-59593-540-1.

Leontiadis, I. y Mascolo, C. (2007). GeOpps: Geographical Opportunistic Routing for Vehicular Networks. En *World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007. IEEE International Symposium on a*, p. 1–6.

Li, F. y Wang, Y. (2007). Routing in vehicular ad hoc networks: A survey. *Vehicular Technology Magazine, IEEE*, **2**(2): 12–22.

Li, T., Li, Y., y Liao, J. (2009). A Contention-Based Routing Protocol for Vehicular Ad Hoc Networks in City Environments. En *Distributed Computing Systems Workshops, 2009. ICDCS Workshops '09. 29th IEEE International Conference on*, p. 482–487.

Lin, G., Noubir, G., y Rajaraman, R. (2004). Mobility models for ad hoc network simulation. En *Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies (INFOCOM 2004)*, Vol. 1, p. 459–463.

Lind, R., Schumacher, R., Reger, R., Olney, R., Yen, H., y Freeman, R. (1998). The network vehicle-a glimpse into the future of mobile multi-media. En *Digital Avionics Systems Conference, 1998. Proceedings., 17th DASC. The AIAA/IEEE/SAE*, Vol. 2, p. I21/1–I21/8 vol.2.

Liu, S. y Cheng, L. (2008). Active Message Oriented Adaptation Middleware for Collaborative Applications in Heterogeneous Environments. En *Communications, 2008. ICC '08. IEEE International Conference on*, p. 1866–1870.

Lochert, C., Hartenstein, H., Tian, J., Fussler, H., Hermann, D., y Mauve, M. (2003). A routing strategy for vehicular ad hoc networks in city environments. En *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE*, p. 156–161.

Lochert, C., Mauve, M., Füssler, H., y Hartenstein, H. (2005). Geographic routing in city scenarios. *SIGMOBILE Mob. Comput. Commun. Rev.*, **9**(1): 69–72.

Lochert, C., Scheuermann, B., Caliskan, M., y Mauve, M. (2007). The feasibility of information dissemination in vehicular ad-hoc networks. En *Fourth Annual Conference on Wireless on Demand Network Systems and Services WONS '07*, p. 92–99.

LTE (2007). Página web whitepaper de Motorola sobre LTE. Recuperado 20 de marzo de: http://www.motorola.com/web/Business/Solutions/Industry%20Solutions/Service%20Providers/Wireless%20Operators/LTE/_Document/Static%20Files/6833_MotDoc_New.pdf.

Mariyasagayam, N., Menouar, H., y Lenardi, M. (2009). An adaptive forwarding mechanism for data dissemination in vehicular networks. En *IEEE Vehicular Networking Conference (VNC)*, p. 1–5.

Marvie, R., Merle, P., y Geib, J. M. (2002). Separation of concerns in modeling distributed component-based architectures. En *Enterprise Distributed Object Computing Conference, 2002. EDOC '02. Proceedings. Sixth International*, p. 144–154.

Mchannel (2009). Página web de Middleware Mchannel. Recuperado 22 de octubre de: http://ants.etse.urv.es/web/software/released-software/software-mchannel.

Montanari, R., Tonti, G., y Stefanelli, C. (2003). Policy-based separation of concerns for dynamic code mobility management. En *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*, p. 82–90.

MyFord Touch (2011). Página web de sistema MyFord Touch de Ford. Recuperado 28 de mayo de: http://corporate.ford.com/news-center/press-releases-detail/pr-myfordtouch-defines-intuitive-31716.

MyLink (2012). Página web de sistema de infoentretenimiento MyLink de Chevi. Recuperado 22 de octubre de: http://www.chevrolet.com/mylink-vehicle-technology.html.

Naumov, V. y Gross, T. (2007). Connectivity-Aware Routing (CAR) in Vehicular Ad-hoc Networks. En *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, p. 1919–1927.

Ngo, H., Shehzad, A., Liaquat, S., Riaz, M., y Lee, S. (2004). Developing Context-Aware Ubiquitous Computing Systems with a Unified Middleware Framework. En L. Yang, M. Guo, G. Gao, y N. Jha, editores, *Embedded and Ubiquitous Computing*, Vol. 3207 de *Lecture Notes in Computer Science*, p. 672–681. Springer Berlin Heidelberg. ISBN 978-3-540-22906-3.

NissanConnectSM (2011). Página web de sistema NissanConnectSM de Nissan. Recuperado 28 de mayo de: http://nissannews.com/en-US/nissan/usa/releases/nissan-unveils-major-advances-in-connected-car-technology.

Nundloll, V., Blair, G. S., y Grace, P. (2009). A component-based approach for (Re)-configurable routing in VANETs. En *Proceedings of the 8th International Workshop on Adaptive and Reflective MIddleware*, Vol. 2 de *ARM '09*, p. 1–6, New York, NY, USA. ACM. ISBN 978-1-60558-850-6.

Ogier, R., Templin, F., y Lewis, M. (2004). RFC 3684: Topology Dissemination Based on Reverse-Path Forwarding. Recuperado 22 de octubre de: http://tools.ietf.org/html/rfc3684.

Ohta, Y., Ohta, T., Kohno, E., y Kakuda, Y. (2011). A Store-Carry-Forward-Based Data Transfer Scheme Using Positions and Moving Direction of Vehicles for VANETs. En *Autonomous Decentralized Systems (ISADS), 2011 10th International Symposium on*, p. 131–138.

Openstreetmap (2005). Página web de proyecto Openstreetmap. Recuperado 22 de octubre de: http://www.openstreetmap.org/.

OSGi (2000). Página web de la especificación de la plataforma OSGi. Recuperado 22 de octubre de: http://www.osgi.org/Specifications/HomePage.

Park, V. D. y Corson, M. S. (1997). A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. En *Proceedings of the INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, INFOCOM '97, p. 1405–, Washington, DC, USA. IEEE Computer Society. ISBN 0-8186-7780-5.

Peddemors, A., Eertink, H., y Niemegeers, I. (2005). Communication context for adaptive mobile applications. En *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, p. 173 – 177.

Pei, G., Gerla, M., y Chen, T.-W. (2000). Fisheye state routing: a routing scheme for ad hoc wireless networks. En *Communications, 2000. ICC 2000. 2000 IEEE International Conference on*, Vol. 1, p. 70–74 vol.1.

Perkins, C., Belding-Royer, E., y Das, S. (2003). RFC 3561: Ad hoc On-Demand Distance Vector. Recuperado 22 de octubre de: http://www.ietf.org/rfc/rfc3561.txt.

Perkins, C. E. y Bhagwat, P. (1994). Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers. *SIGCOMM Comput. Commun. Rev.*, **24**(4): 234–244.

Popovici, A., Frei, A., y Alonso, G. (2003). A Proactive Middleware Platform for Mobile Computing. En M. Endler y D. Schmidt, editores, *Middleware 2003*, Vol. 2672 de *Lecture Notes in Computer Science*, p. 455–473. Springer Berlin Heidelberg. ISBN 978-3-540-40317-3.

Popovici, D., Desertot, M., Lecomte, S., y Peon, N. (2011). Context-Aware Transportation Services (CATS) Framework for Mobile Environments. *IJNGC*, **2**(1).

Pour, G. (1998). Moving toward component-based software development approach. En *Technology of Object-Oriented Languages, 1998. TOOLS 27. Proceedings*, p. 296–300.

Qian, Y. y Moayeri, N. (2008). Design of Secure and Application-Oriented VANETs. En *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, p. 2794–2799.

R-Link (2011). Página web de sistema R-Link de Renault. Recuperado 21 de marzo de: http://www.renault.com/en/innovation/plaisir-et-confort/pages/r-link.aspx.

Ramasubramanian, V., Haas, Z. J., y Sirer, E. G. (2003). SHARP: a hybrid adaptive routing protocol for mobile ad hoc networks. En *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, MobiHoc '03, p. 303–314, New York, NY, USA. ACM. ISBN 1-58113-684-6.

Ramdhany, R., Grace, P., Coulson, G., y Hutchison, D. (2009). MANETKit: supporting the dynamic deployment and reconfiguration of ad-hoc routing protocols. En *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '09, p. 1:1–1:20, New York, NY, USA. Springer-Verlag New York, Inc.

Reichardt, D., Miglietta, M., Moretti, L., Morsink, P., y Schulz, W. (2002). CarTALK 2000: safe and comfortable driving based upon inter-vehicle-communication. En *Intelligent Vehicle Symposium, 2002. IEEE*, Vol. 2, p. 545–550.

Rejaie, R., Handley, M., y Estrin, D. (1999). Quality adaptation for congestion controlled video playback over the Internet. *SIGCOMM Comput. Commun. Rev.*, **29**(4): 189–200.

Royer, E. M. y Perkins, C. E. (1999). Multicast operation of the ad-hoc on-demand distance vector routing protocol. En *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, MobiCom '99, p. 207–218, New York, NY, USA. ACM. ISBN 1-58113-142-9.

Safi, Q. G. K., Nawaz, T., Shah, S. M. A., y Mahmood, T. (2011). Intelligent Device Independent UI Adaption for Heterogeneous Ubiquitous Environments. *IJCSNS International Journal of Computer Science and Network Security*, **11**(11): 5–79.

Salber, D., Dey, A. K., y Abowd, G. D. (1999). The context toolkit: aiding the development of context-enabled applications. En *Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, CHI '99, p. 434–441, New York, NY, USA. ACM. ISBN 0-201-48559-1.

Schnaufer, S. y Effelsberg, W. (2008). Position-based unicast routing for city scenarios. En *World of Wireless, Mobile and Multimedia Networks, 2008. WoWMoM 2008. 2008 International Symposium on a*, p. 1–8.

Schoch, E., Kargl, F., Weber, M., y Leinmuller, T. (2008). Communication patterns in VANETs. *Communications Magazine, IEEE*, **46**(11): 119–125.

Shen, C.-C., Huang, Z., y Jaikaeo, C. (2006). Directional Broadcast for Mobile Ad Hoc Networks with Percolation Theory. *IEEE Transactions on Mobile Computing*, **5**(4): 317–332.

Shladover, S. y Tan, S.-K. (2006). Analysis of Vehicle Positioning Accuracy Requirements for Communication-Based Cooperative Collision Warning. *Journal Of Intelligent Transportation Systems*, **10**(3): 131–140.

Singh, J., Bambos, N., Srinivasan, B., Clawin, D., y Yan, Y. (2003). Proposal and demonstration of link connectivity assessment based enhancements to routing in mobile ad-hoc networks. En *Vehicular Technology Conference, 2003. VTC 2003-Fall. 2003 IEEE 58th*, Vol. 5, p. 2834–2838 Vol.5.

Soares, R., Nakamura, E., Figueiredo, C., y Loureiro, A. A. F. (2012). VCARP: Vehicular Ad-hoc Networks context-aware routing protocol. En *Computers and Communications (ISCC), 2012 IEEE Symposium on*, p. 442–447.

Su, W., Lee, S.-J., y Gerla, M. (2001). Mobility prediction and routing in ad hoc wireless networks. *Int. J. Netw. Manag.*, **11**(1): 3–30.

Sun, B., Wu, H., Zhao, H., y Hu, X. (2010). Research and application on plug-in technology in OpenSim. En *Audio Language and Image Processing (ICALIP), 2010 International Conference on*, p. 1392–1396.

Sunderam, V. y Kurzyniec, D. (2002). Lightweight self-organizing frameworks for metacomputing. En *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, p. 113–122.

SYNC (2007). Página web de sistema SYNC de Ford. Recuperado 21 de marzo de: http://www.ford.com/technology/sync/.

Taliwal, V., Jiang, D., Mangold, H., Chen, C., y Sengupta, R. (2004). Empirical determination of channel characteristics for DSRC vehicle-to-vehicle communication. En *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*, VANET '04, p. 88–88, New York, NY, USA. ACM. ISBN 1-58113-922-5.

Tian, J., Han, L., y Rothermel, K. (2003). Spatially aware packet routing for mobile ad hoc inter-vehicle radio networks. En *Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE*, Vol. 2, p. 1546–1551 vol.2.

Tizen (2012). Página web de Tizen, proyecto opensource para futuros sistemas vehiculares. Recuperado 26 de Octubre de: https://source.tizen.org/.

TomHardware1 (2012). Página web donde se indica las especificciones de hardware del sistema vehicular MyFordTouch. Recuperado 29 de septiembre de: http://www.tomshardware.com/reviews/ford-focus-infotainment-sync-myford,3206-2.html.

TomHardware2 (2013). Página web donde se indica las especificciones de hardware del sistema vehicular Uconnect Access de Jeep. Recuperado 26 de septiembre de: http://www.tomshardware.com/reviews/jeep-grand-cherokee-uconnect-access-review,3573-3.html.

Uconnect (2013). Página web del sistema vehicular se indica las especificciones Uconnect Access de Jeep. Recuperado 26 de septiembre de: http://www.driveuconnect.com/features/uconnect_access/, http://www.driveuconnect.com/features/uconnect_access/.

Unik-olsr (2005). Página web de implementacion de protocolo OLSR Unik-olsr. Recuperado 29 de mayo de: http://www.olsr.org/.

VICS (1990). Página web de proyecto VICS. Recuperado 19 de Marzo de: http://www.vics.or.jp/english/vics/index.html.

Wang, B., Yin, D., Wu, T., y Sheng, J. (2011). Plug-In Based Integrated Development Platform for Industrial Control System-P-IDP4ICS. En *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*, p. 1578–1583.

Wang, X., Dong, J., Chin, C., Hettiarachchi, S., y Zhang, D. (2004). Semantic Space: an infrastructure for smart spaces. *Pervasive Computing, IEEE*, **3**(3): 32–39.

WiMAX (2001). Página web de WiMAX Forum. Recuperado 20 de marzo de: http://www.wimaxforum.org/index.htm.

Win, B. D., Piessens, F., Joosen, W., y Verhanneman, T. (2003). On the Importance of the Separation-of-Concerns Principle in Secure Software Engineering. En *In ACSA*

Workshop on the Application of Engineering Principles to System Security Design - Final Report (Serban, C., ed.), p. 1–10.

Wired (2012). Página web de noticia sobre actualizacion por el aire del software del vehiculo Tesla modelo S. Recuperado 28 de mayo de: http://www.wired.com/autopia/2012/09/tesla-over-the-air.

Wischhof, L., Ebner, A., Rohling, H., Lott, M., y Halfmann, R. (2003). Adaptive broadcast for travel and traffic information distribution based on inter-vehicle communication. En *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE*, p. 6–11.

Wischhof, L., Ebner, A., y Rohling, H. (2005). Information dissemination in self-organizing intervehicle networks. *Intelligent Transportation Systems, IEEE Transactions on*, **6**(1): 90–101.

Wolfinger, R., Dhungana, D., Prähofer, H., y Mössenböck, H. (2006). A Component Plug-In Architecture for the .NET Platform. En D. Lightfoot y C. Szyperski, editores, *Modular Programming Languages*, Vol. 4228 de *Lecture Notes in Computer Science*, p. 287–305. Springer Berlin / Heidelberg. 10.1007/1186099018.

Wu, H., Fujimoto, R., Guensler, R., y Hunter, M. (2004). MDDV: a mobility-centric data dissemination algorithm for vehicular networks. En *Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*, VANET '04, p. 47–56, New York, NY, USA. ACM. ISBN 1-58113-922-5.

Yau, S., Karim, F., Wang, Y., Wang, B., y Gupta, S. (2002). Reconfigurable context-sensitive middleware for pervasive computing. *Pervasive Computing, IEEE*, **1**(3): 33 –40.

Yawut, C., Paillassa, B., y Dhaou, R. (2008). Mobility Metrics Evaluation for Self-Adaptive Protocols. *Journal of Networks*, **3**(1): 53–64.

Yi, Y., Lee, S.-J., Su, W., y Gerla, M. (2002). IETF Internet Draft: On-Demand Multicast Routing Protocol (ODMRP) for Ad Hoc Networks. Recuperado 22 de octubre de: http://tools.ietf.org/id/draft-ietf-manet-odmrp-04.txt.

Yin, J., Holland, G., ElBatt, T., Bai, F., y Krishnan, H. (2006). DSRC Channel Fading Analysis from Empirical Measurement. En *Communications and Networking in China, 2006. ChinaCom '06. First International Conference on*, p. 1–5.

Yousefi, S., Mousavi, M., y Fathy, M. (2006). Vehicular Ad Hoc Networks (VANETs): Challenges and Perspectives. En *6th International Conference on ITS Telecommunications Proceedings*, p. 761–766.

Zhao, J. y Cao, G. (2006). VADD: Vehicle-Assisted Data Delivery in Vehicular Ad Hoc Networks. En *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, p. 1 –12.

Apéndice A

Prototipo de la arquitectura PLUGAM

Como parte de este trabajo de tesis se decidió implementar como prueba de concepto, un prototipo del *middleware* PLUGAM. En este capítulo se presenta los detalles de implementación del prototipo. Un aspecto importante del desarrollo de este prototipo es el encontrar y elegir una plataforma de *plug-in*. Además, el trabajar con una implementación de la arquitectura PLUGAM, ayuda a definir con detalles de bajo nivel de la arquitectura al saltar de la teoría a la práctica, y por ende tener una propuesta más robusta.

Con respecto a la selección de una plataforma de *plug-in* adecuada, se encontraron varios *frameworks* y plataformas de *plug-in* en los lenguajes de programación de Java, C# y HTML. Por ejemplo, el Extensiones de Chrome (2012) permite la creación de extensiones (sinónimo de *plug-ins*) desarrollado en HTML y empaquetados dentro de archivos zip, los cuales contiene el código HTML y un archivo de manifiesto en formato JSON con la meta-información de la extensión.

En el caso de lenguajes de programación más tradicionales como C# y Java, en C# el código del *plug-in* es empaquetado en un archivo binario DLL y en Java como un archivo JAR. Los meta-datos del *plug-in*, utilizados para identificar y administrar al mismo, son representados en C# como atributos y servicios de reflexión de .NET, en Java son representados como un archivo de texto manifiesto introducido dentro del archivo JAR.

Volviendo a la descripción de la plataforma de *plug-ins* del prototipo, se utilizaron las tecnologías base a OSGi (2000) y iPOJO (2008) (ver la figura 32).

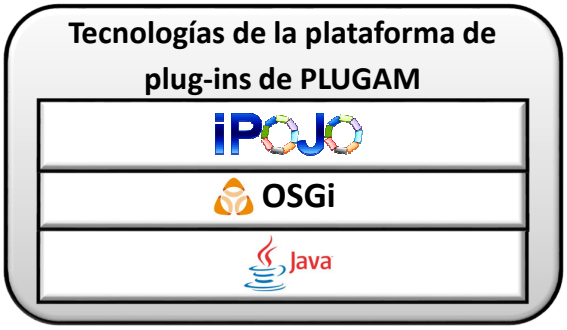

Figura 32. Tecnologías utilizadas en la plataforma de plug-ins del prototipo

La especificación OSGi es una arquitectura basada en el lenguaje de programación Java, para el desarrollo de aplicaciones modulares, estos módulos son representados como archivos JAR los cuales son llamados *bundles* dentro de OSGi. Relacionando con la arquitectura PLUGAM, un *plug-in* de PLUGAM es representado por un *bundle* que ofrece los servicios del *plug-in* al entorno OSGi; el *middleware* de adaptación es representado por otro *bundle* que consume los servicios de los *plug-ins* disponibles en el entorno OSGi.

El motivo principal de seleccionar OSGi, es que es una plataforma robusta y completa, su comunidad de desarrollo es muy activa actualmente, existe mucho esfuerzo de la comunidad por mejorar y extender la plataforma, y además cuenta con muchas herramientas de terceros para manejar los *bundles*.

La otra tecnología base de la plataforma *plug-in* del prototipo es iPOJO (2008), el propósito de esta tecnología es el simplificar el desarrollo de aplicaciones OSGi y

promover la programación orientada a servicios.

Basado en el concepto de POJO (que significa *"Plain Old Java Object"*), el objetivo de iPOJO es que la lógica de la aplicación se desarrolle con mayor facilidad. Gracias a iPOJO no existe necesidad de hacer referencia o hacer llamadas al código OSGi al momento de implementar el módulo de la aplicación utilizando OSGi. Además la vinculación de la lógica de aplicación con el componente iPOJO y un *bundle* OSGi, es realizada a nivel de compilación. Dicho de otra manera, el uso de la tecnología iPOJO reduce el desarrollo de un *plug-in* en el prototipo a simplemente escribir una clase que extiende directamente de Object y que implementa una interfaz Java, siendo esta una las interfaces de *plug-in* de PLUGAM; lo anterior permite un ahorro en tiempo y esfuerzo de desarrollo al implementar un *plug-in*.

Por otro lado, la especificación de la plataforma OSGi tiene en la actualidad varias implementaciones de código abierto conocidas como contenedores OSGi; por ejemplo los contenedores Equinox (2005), Apache Felix (2006) y Knopflerfish (2005). En la definición del prototipo se eligió utilizar el contenedor OSGi Apache Felix (2006), la razón es su grado de desarrollo y que la comunidad está muy activa hoy en día, además es utilizado como base para arquitecturas de calidad industrial (por ejemplo en Apache ServiceMix (2008)), por lo que es robusto.

El *middleware* PLUGAM se implementó como un *bundle* OSGi, el cual corre en el mismo contenedor OSGi que los *plug-ins*. Al momento que el usuario inicia el *bundle* del *middleware*, este detecta todos los *plug-ins* que fueron cargados en el contenedor OSGi (ver figura 33). Además de definir la plataforma *plug-in* del prototipo, también se

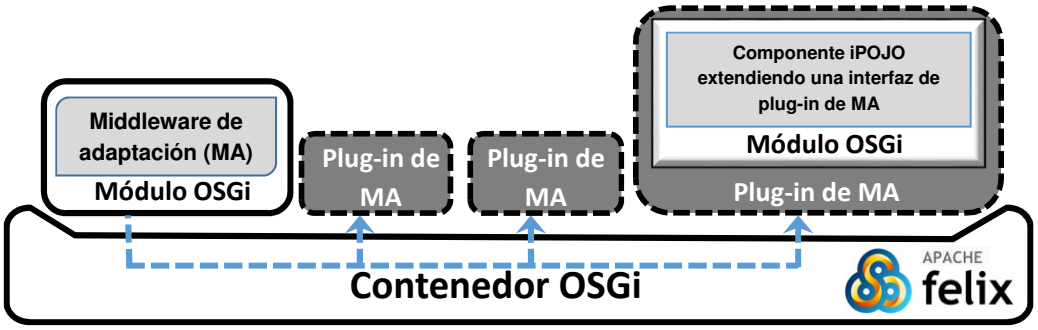

Figura 33. Descripción gráfica de relación entre el middleware y plug-ins del prototipo PLUGAM en Apache Felix

implementó algunos *plug-ins* de prueba; específicamente aquellos necesarios para poder implementar en el prototipo el caso de estudio 1, los cuales son los siguientes:

- *Plug-in* tipo protocolo conteniendo a OLSR: Éste es desarrollado utilizando el código en JAVA de una implementación de OLSR, el cual es extraído del código fuente del proyecto *middleware* Mchannel (2009). Este *plug-in* ofrece además una acción de adaptación la cual puede modificar la constante (ahora transformada en variable) HELLO_INTERVAL de OLSR.
- *Plug-in* tipo contextE ofreciendo información de un GPS: Este *plug-in* encapsula a un módulo que ofrece como elementos de contexto cierta información arrojada por un dispositivo GPS mediante conexión inalámbrica Bluetooth. Este *plug-in* ofrece los siguientes elementos de contextos: velocidad del nodo en base a dos posiciones GPS, velocidad del nodo calculado internamente por el GPS utilizando el efecto *Doopler* de la señal proveniente del satélite (obtenido de las lecturas tipo NMEA), longitud de la posición GPS, latitud de la posición GPS, tiempo actual, número de satélites detectados por el GPS. La implementación de este *plug-in* consta de la utilización y modificación de parte del código fuente del proyecto

GPSylon (2009).

- Plug-in tipo algoritmo: La implementación de este *plug-in* es la simple codificación del la asignación directa entre la velocidad del vehículo y el nuevo HELLO_INTERVAL descrita en la sección 6.1..

A.1 Desarrollo plug-ins en el prototipo

Volviendo a iPOJO, para ilustrar la facilidad que otorga al desarrollo de un *plug-in* PLUGAM, a continuación se describe el proceso a seguir para implementar un *plug-in* utilizando la plataforma de programación Eclipse. El proceso es el siguiente:

- Crear un nuevo proyecto Java en Eclipse, después importar la librería PluginServices.jar en el proyecto. Este archivo jar es un *bundle* OSGi que contiene todas las interfaces de los *plug-ins* y clases auxiliares, necesarias para la interacción *plug-in* y *middleware*.
- El siguiente paso es crear una clase Java en el proyecto, después hacer que esta implemente una de las interfaces de los tipos de *plug-in* PLUGAM.
- Fuera de Eclipse y en el mismo directorio que los archivos del proyecto, se agregan y personalizan los siguientes archivos: build.xml, metadata.xml y <Nombre proyecto>.bnd. El desarrollador del *plug-in* tendrá acceso a ejemplos de estos archivos y estos se personalizan dependiendo del proyecto y meta-información del *plug-in*.
- Fuera de Eclipse, se compila el proyecto utilizando Apache Ant y los archivos del paso anterior. Después de la compilación, se crea un archivo jar conteniendo al *plug-in* encapsulado en el componente iPOJO y un *bundle* OSGi.

- Por último, el archivo jar resultante se coloca en el folder de *plug-ins* dentro del directorio de Apache Felix.

A.2 Instalación y ejecución del prototipo PLUGAM

Por otro lado, la instalación del prototipo en un sistema de cómputo es bastante sencilla gracias a Java y Apache Felix, involucra simplemente copiar el contenido del directorio de Apache Felix modificado para PLUGAM (modificado ya que contiene los *bundles* adicionales del *middleware* PLUGAM). Paso siguiente, se ponen los *plug-ins* creados por uno mismo u obtenido de una tercera persona, dentro del folder predeterminado para *plug-ins* dentro de Apache Felix.

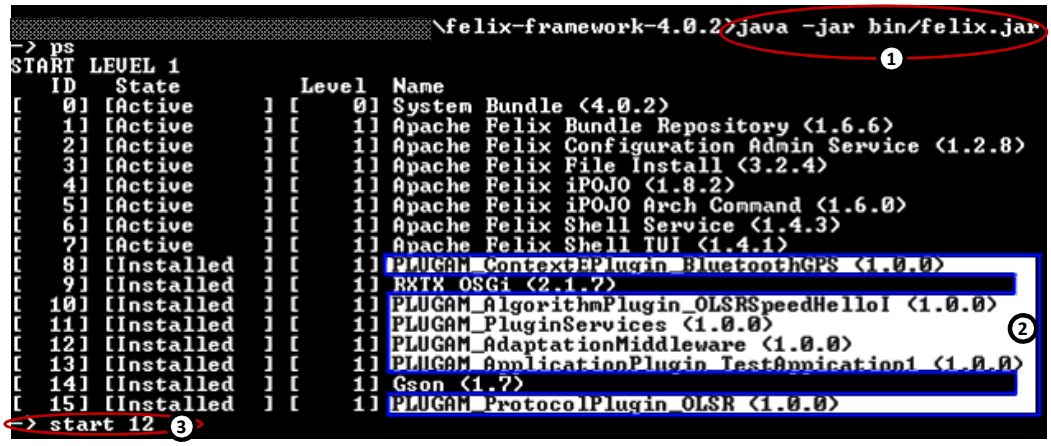

Figura 34. Apache Felix corriendo el bundle del middleware de adaptacion (1), ademas se muestran en azul (2) el bundle pluginServices y los plug-ins del caso de estudio 1

En cuanto a la ejecución del prototipo PLUGAM, primero se debe ejecutar el archivo jar conteniendo el ejecutable de Apache Felix (como se muestra en (1) de la figura 34), después esperar algunos segundos hasta que Apache Felix detecte los *plug-ins* que se instalaron, y por último empezar a ejecutar el *middleware* escribiendo el comando

"start" dentro de la consola de Apache Felix (como se muestra en (3) de la figura 34). En (2) de la figura 34 se pueden ver los bundles involucrados en la implementación del caso de estudio 1 en el prototipo.