IEEE TRANSACTIONS ON
EMERGING TOPICS
IN COMPUTING

Received 8 July 2014; revised 8 November 2014; accepted 6 January 2015. Date of publication 21 January, 2015; date of current version 6 March, 2015.

Digital Object Identifier 10.1109/TETC.2015.2395959

S-Aframe: Agent-Based Multilayer Framework With Context-Aware Semantic Service for Vehicular Social Networks

XIPING HU[1], JIDI ZHAO[2,3], (Member, IEEE), BOON-CHONG SEET[4], (Senior Member, IEEE), VICTOR C. M. LEUNG[1], (Fellow, IEEE), TERRY H. S. CHU[5], AND HENRY CHAN[5], (Member, IEEE)

[1]Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada
[2]Institute of System Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
[3]Department of Public Administration, East China Normal University, Shanghai 200062, China
[4]Department of Electrical and Electronic Engineering, Auckland University of Technology, Auckland 1010, New Zealand
[5]Department of Computing, The Hong Kong Polytechnic University, Hong Kong

CORRESPONDING AUTHOR: X. HU (xipingh@ece.ubc.ca)

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the NSERC DIVA Strategic Network, in part by TELUS and other industry partners, in part by the National Science Foundation of China under Grant 71001068, in part by the Shanghai Pujiang Program, in part by the ECNU International Publication Grant, in part by SJTU Post-Doctoral Initial Grant, and in part by the Department of Computing, The Hong Kong Polytechnic University, Hong Kong.

ABSTRACT This paper presents S-Aframe, an agent-based multilayer framework with context-aware semantic service (CSS) to support the development and deployment of context-aware applications for vehicular social networks (VSNs) formed by in-vehicle or mobile devices used by drivers, passengers, and pedestrians. The programming model of the framework incorporates features that support collaborations between mobile agents to provide communication services on behalf of owner applications, and service (or resident) agents to provide application services on mobile devices. Using this model, different self-adaptive applications and services for VSNs can be effectively developed and deployed. Built on top of the mobile devices' operating systems, the framework architecture consists of framework service layer, software agent layer and owner application layer. Integrated with the proposed novel CSS, applications developed on the framework can autonomously and intelligently self-adapt to rapidly changing network connectivity and dynamic contexts of VSN users. A practical implementation and experimental evaluations of S-Aframe are presented to demonstrate its reliability and efficiency in terms of computation and communication performance on popular mobile devices. In addition, a VSN-based smart ride application is developed to demonstrate the functionality and practical usefulness of S-Aframe.

INDEX TERMS Vehicular social networks, software agent, context-aware semantic service, application development.

I. INTRODUCTION

According to an investigation by the U.S. Department of Transportation, personal vehicles account for 21% of the global energy consumption, while the average occupancy rate of personal vehicle trips is 1.6 persons per vehicle-mile [1]. Given that each personal vehicle can commonly carry up to 4 persons, it means that 60% of the carriage capacity is wasted during personal trips. On the other hand, on-road traffic is reported to be one of the major sources of air pollution and thick hazes in many large cities on earth. In order to improve transportation efficiency and environment protection, researchers have proposed and developed rideshare applications [2], [3]. However, most of them lack real-time social interactions or exchange of context information between the drivers and potential passengers. Thus, it is difficult for each party to analyze which vehicle/passenger can best match their needs and preferences. Consequently, there is an emerging need to form recurring virtual mobile

communication networks and social communities between these travelers or their vehicles in the form of vehicular social networks (VSNs) [4], so as to facilitate the effective and efficient interactions between them. Furthermore, previous research has shown that knowledge from the social interactions between nodes can help to improve the performance of mobile systems [5]. Therefore, it is anticipated that beyond the transportation scenarios mentioned above, VSN applications can be effectively used for many other purposes. The following three types of applications are anticipated over VSNs: (a) Safety improvements: applications that improve the safety of drivers, passengers and pedestrians by notifying them about hazardous road situations [6]; (b) Traffic management: applications that provide users with up-to-date traffic information and recommendations that enable them to make better route decisions to reduce travel time and expense; (c) Entertainment: applications that enable the streaming, downloading, or sharing of multimedia files [7].

VSN systems are built on top of vehicular networks that provide connectivity between users and devices participating in the VSN as well as the Internet at-large. While cellular networks can provide such connectivity, the cost may be too high and the latency too large. Instead, a vehicular ad-hoc network (VANET) may be established to connect the users and devices onboard vehicles that are physically close to each other, e.g., traveling on the same street or stretch of highway. This is feasible because contemporary mobile devices such as smart phones and tablets, which people normally carry with them onboard vehicles, have the capability to establish opportunistic ad-hoc networks through WiFi Direct or Bluetooth connectivity [8]. Thus, in-vehicle and road-side users who are close-by can use their mobile devices to *inexpensively* establish a VANET without the need for accessing the cellular infrastructure [9]. A VANET only provides the underlying networking connectivity between the road users. A new breed of VSN applications that could leverage or adapt to the opportunistic nature and characteristics of VANETs is still needed, as existing popular social network applications (e.g., Facebook) do not support local ad-hoc social networking but require a persistent Internet connection that may not be available in VANETs. The objective of this paper is to present a novel software platform that facilitates the development and deployment of VSN applications and services over VANETs that may for formed opportunistically.

Due to the dynamism of VANETs and the opportunism of user connections in a VSN, dynamic network connectivity (e.g., as vehicles move at high speeds, the wireless links may become unreliable and have short lifetimes) and users' dynamic contexts of VSN (which may vary with changing VSN context, such as user location, time periods, identities of objects, etc.) are unique and important challenges faced by VSN applications [10]. Middleware is a layer of software that manages the interactions between applications and the underlying network by providing various services to the applications, which can simplify the application development process. However, existing middlewares for VANETs depend on specific hardwares and have only been designed for specific applications [11], or they strictly constrains the behaviors of applications in undertaking some simple tasks and are not scalable or extensible to undertake more complicated tasks [12]. They also lack the abilities of autonomous self-management and self-adaptivity to users' dynamic contexts within VSNs. Thus, they are inadequate for supporting different VSN applications that need the capability of self-adaptation upon VANETs [13].

Consequently, it is of interest to investigate and develop a software platform that uses the high-level application programming approach to facilitate the development and deployment of a diverse range of VSN applications. This platform should incorporate services that enable applications to self-adapt to dynamic network connectivity and users' dynamic contexts within VSNs during runtime. To the best of our knowledge, such a platform does not exist currently. This work fills the gaps identified above by presenting an effective software platform for ubiquitous VSN applications.

The primary contribution of this paper is the proposal, design and implementation of S-Aframe, which is the first mobile software platform that supports real-world development and deployment of VSN applications on Andriod devices. S-Aframe hides the complexity of handling changing network connectivity, and varying basic user services requirements of VSNs, by providing a high-level multi-agent programming model with extensibility support. Also, S-Aframe provides a rich set of framework services with a standard Java application programming interface (API) format. Therefore, S-Aframe provides a modeling paradigm to facilitate the development and deployment of different context-aware mobile applications for VSNs.

Our second contribution is the proposal of a new solution called context-aware semantic service (CSS) that integrates software agent and semantic techniques in S-Aframe to effectively manage and utilize various context information for different VSN applications. CSS can enable agent based applications developed on S-Aframe to intelligently determine what and how services and information should be delivered to users, and autonomously adapt the behaviors of these services to users' dynamic contexts during run-time, e.g., automatically assist drivers to analyze and accept/reject rideshare requests, as they can hardly use mobile devices while driving.

The rest of the paper is organized as follows. Section 2 reviews related research on VSNs. Section 3 introduces the programming model and features of S-Aframe. Section 4 presents the design and implementation approaches of S-Aframe, and discusses how S-Aframe satisfies the key requirements of VSN applications. Section 5 evaluates S-Aframe through a set of practical experiments. Section 6 presents *smart ride*, a practical VSN ridesharing application developed based on S-Aframe. Section 7 concludes the paper.

II. RELATED WORKS

In this section, we review the related works in the literature, which inspire the design of S-Aframe with CSS, and discuss the difference between them and our approach.

RoadSpeak [11] is the first framework proposed for VSNs, which allows commuters to automatically join voice chat groups on roadways. Unlike traditional social networks, RoadSpeak considers, in addition to the interests of users, the time intervals and locations in its definition of the VSN profile when user groups are formed. RoadSpeak partially supports extensibility. It provides a number of Java APIs to application developers, based on which developers can extend RoadSpeak clients to provide enhanced functionality. Nevertheless, RoadSpeak relies on client-server interactions. In a VANET environment, it is difficult to provide a stable server among the vehicles all the time. In addition, RoadSpeak only provides a voice chat service. Its extension support is merely for the grouping of membership in this service, and not for application developers to extend it to provide other services and functions for VSNs. Thus, it can hardly fulfill the diverse and ubiquitous service requirements of different users in a VSN.

Recently, several semantic based service systems over dynamic networks have been proposed [14], [15]. Fujii et al. proposed a semantic-based context-aware dynamic service composition framework (SCDSCF) [14] using the semantics of components and user contexts over the World Wide Web. SCDSCF consists of a strict abstract model for modeling semantics, a supporting middleware, and a service composition mechanism. However, it models the elements in each component, including the semantics, contexts as well as the user, in a strict manner, which results in significant efforts for programmers to follow for further application development. It also causes much redundancy in information representations that may affect its efficiency. The framework may offer a generic approach for the Web, but is not necessarily suitable in the context of VANETs. MobiSN [15] is a framework proposed for social network construction with mobile phones, which incorporates a semantic-based matching method that works with user profiles. The framework only addresses a specific application domain, i.e., social grouping, and does not provide a generic approach for a range of applications. MobiSN also lacks support for interoperability among distributed elements and does not consider adaptation of applications to changing contexts in dynamic environments; thus it could hardly be used for VSN applications.

AmbientTalk [16] is an experimental programming language for developing mobile peer-to-peer applications. A feature of its programming paradigm is that it accounts for the possibility of network failures in its programming model. In addition, it employs a purely event-driven concurrency framework based on actors. In AmbientTalk, actor executions can be concurrent with asynchronous actor method invocations; thus AmbientTalk is very suitable for highly dynamic networks. However, AmbientTalk is a completely new language, which means that programmers have to spend considerable time to become familiar with it before using it to develop applications for VSN systems. Furthermore, AmbientTalk does not provide any key services to handle the unique challenges of VSN applications, thus handicapping the application development of VSNs. In the implementation of S-Aframe, we encapsulate AmbientTalk as new Java libraries by making use of its symbiotic programming mechanism, and integrate new generic services into its multilayer framework with an agent-based programming model. Consequently, S-Aframe extends AmbientTalk by retaining its advantages, while further providing a modeling paradigm with CSS to facilitate the development and deployment of context-aware mobile applications for VSNs using Java.

III. MODEL

The programming model of S-Aframe is shown in Figure 1. It is designed to facilitate the development and deployment of context-aware VSN applications on mobile devices over VANETs. A software agent can be seen as a composition of software and data, which is able to function autonomously. Mobile agents are software agents that act on behalf of their creators and can function without a continuous active connection; e.g., after they have migrated to a node, even if the connection is broken, they can autonomously proceed to process the data, and return the results and/or migrate to the next node when the connection resumes. Thus, mobile agents are well-suited for executing applications in the dynamic network environment of VANETs. Moreover, mobile agents can be adopted to realize savings in terms of network traffic load by transmitting the agent code instead of raw data over VANETs. Consequently, an agent based programming model is adopted in S-Aframe as shown in Figure 1.

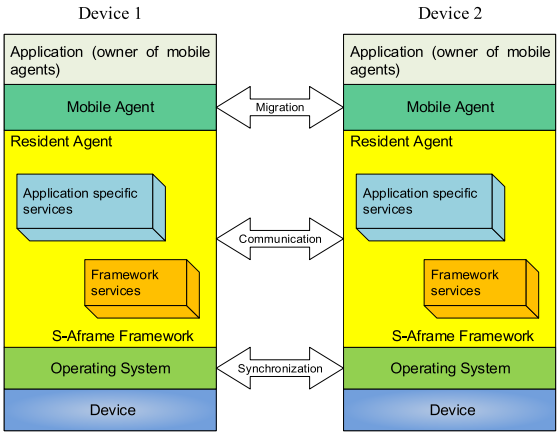


FIGURE 1. Programming model of S-Aframe.

A. SCALABILITY

Conventional agent systems are usually designed for specific applications. In systems such as Agilla [12], mobile agents contain all the relevant codes when they want to accomplish certain tasks, although some of the codes may never be used,

resulting in wasted network resources. On the other hand, this approach requires developers to develop a completely new mobile agent whenever a new application of VSN is to be implemented; this is so even when the new application shares common features with existing applications and use similar application services. Consequently, developers have to spend a significant amount of time developing different applications for VSNs, resulting in a low efficiency in application development. Thus, in S-Aframe, we adopt a novel type of agents - resident agents as service agents to provide a reusable set of application services to mobile agents. Also, using resident agents to provide application services to mobile agents via agent communications not only enables more flexible operations of the mobile agents, but also provides scalable options to different mobile devices (e.g., on-demand deployment of resident agents and services) as the hardware constraints of them are heterogeneous. Note that the former agent-based solutions in mobile computing [17] usually provide services to mobile agents through a hosting middleware/general agent.

In S-Aframe, based on the programming model, an application developer could develop resident agents, mobile agents, and applications. Resident agents provide application services on mobile devices of VSN systems. Such application services provide local resources for the application to visiting mobile agents. To program resident agents, the programmer can invoke, configure and extend a common set of application services provided by S-Aframe. Mobile agents are created by applications and they can automatically migrate around VSNs with their states and execution results. They can dynamically use different application services provided by resident agents on local nodes to accomplish specific application tasks. An application, as the owner of a mobile agent, can send the mobile agent to execute application services in an underlying VANET and retrieve it back to the mobile node running the application.

B. ADAPTATION TO DYNAMIC NETWORK CONNECTIVITY

Since the network connectivity of the underlying VANET is dynamic and frequently changes, each mobile node can join or leave the network anytime and anywhere, which may cause VSN application failures. For example, without knowing the failures of targeting nodes when mobile agents execute tasks around VANETs, they may get lost in such networks and fail to bring the expected results back to the users, while the lack of sufficient services and resource in a new mobile node joining a VSN may also impact the correctness of the execution results of mobile agents for VSN applications. Thus, S-Aframe provides self-healing and self-configuring capabilities [18] over dynamic network connections as key framework services to VSN applications. In S-Aframe, for VSN applications to self-heal, it enables the resident agents deployed on every mobile node to continue monitoring the status of a VANET (e.g., which nodes have just become unavailable in a VSN), and provide the information to the mobile agents to help them to adapt to node or link failures. Also, for self-configuration, S-Aframe provides a service that can dynamically and automatically deploy the core functions and services to a new mobile node when it joins a VSN upon VANETs, and if a mobile device does not have the necessary resources or services, mobile agents can also automatically transfer the necessary resources or services between mobile nodes.

C. ADAPTATION TO USERS' DYNAMIC CONTEXTS

In a VSN system, because of the opportunism of user connections, the changing contexts of the users may also result in users' dynamic contexts. However, traditionally, the descriptive information of the service requester is compared to that of the service provider, and their similarity is measured using traditional service matching by simple string-matching (e.g., location based [12], identities based [19]) methods. Such an approach cannot work well as it is not realistic to require service requesters and service providers to use exactly the same contexts (e.g., words about their destinations in a rideshare application) in open and dynamically changing environments of VSNs. Therefore, semantic techniques and context information models are incorporated in CSS as a key framework service of S-Aframe, to enable VSN applications to dynamically and intelligently adapt to different practical VSN scenarios, e.g., matching users' requirements and accessible services, automatically finding services and information to meet users' requirements, recommending nearby VSN users with similar destinations/preferences, and so on. More specifically, in order to adapt to users' dynamic contexts of VSNs, CSS can enable the resident agents of VSN applications to monitor any change in user contexts while the mobile agents are executing the requested service. Upon detecting any changes, CSS can intelligently compose a new service instance that better suits the user's new contexts and provide it to the mobile agents through the resident agents. In order to adapt to different users, in addition to learning users' preferences in different contexts from historical information, S-Aframe also allows users to explicitly specify rules for services they prefer in a specific context.

D. IMPACT AND COST

Considering the limited resource of mobile devices (i.e., computing power, storage capacity, and battery energy), unstable networking connections of VANETs, and time efficiency requirements of most VSN application scenarios, the impacts and costs of mobile VSN applications on mobile devices are always a concern. Thus, we adopt weak adaptation [12] as the primary option in S-Aframe, so as to decrease the resource requirements on mobile devices and ensure that VSN applications developed on S-Aframe have considerable time efficiency in finishing tasks. Also, in order to reduce networking overhead of VSN systems, S-Aframe divides data into two types: shared and non-shared. The shared data contain only the basic context information (e.g., ID name and available services) of the local device, which will be shared with the VSN system by the resident agents. On the other hand, non-shared data contain the agent codes, application

specific services, existing data in local mobile devices, and execution results of mobile agents, etc. In addition, when developers develop mobile agents, they can decide whether the mobile agents should store the processing results locally in the mobile devices, and what data mobile agents should carry over to the next devices to which they migrate.

E. SECURITY AND PRIVACY

Security and privacy are common concerns for mobile applications over VSNs. Firstly, a malicious user can masquerade as a legitimate user by intercepting and replaying his/her identity to other mobile devices in VSNs. The dynamic security framework proposed in [20] addresses this problem by minimizing the impacts of malicious behaviors in VSNs. Using this framework, a model was designed to derive the probabilities of admitting and trusting a malicious node in VSNs. Furthermore, trust is another important aspect for VSN applications, since it enables entities to cope with uncertainty and uncontrollability caused by the free will of others in VSNs. The vehicular network trust model proposed in [21] integrates cryptography-based entity trust, which provides security protections on data origin and integrity, and social trust, which provides a level of belief in the transmitted data. A trust architecture that also models situation-aware trust to address several important privacy issues in VSNs is presented in [22]. In addition, authentication and access control are vital for security and privacy of VSN applications given the open environment and the possible presence of threats where adversaries can monitor and expose sensitive data of other participants. There are several existing solutions for addressing these issues. For instance, a combined authentication-based multilevel access control to preserve privacy in collaborative environments is proposed in [23], while a lightweight mutual authentication mechanism for network mobility is presented in [24]. In [25], several complementary security mechanisms of public key infrastructure (PKI) for VANETs are presented. Even though the current version of S-Aframe does not incorporate the security mechanisms mentioned above, these mechanisms can be integrated into S-Aframe as a framework service to achieve different levels of security.

IV. DESIGN AND IMPLEMENTATION OF S-AFRAME

As proposed in Section 3, S-Aframe is designed to provide a systematic approach to facilitate the development and deployment of context-aware VSN applications, particularly applications that self-adapt to dynamically changing VANET environments and users' dynamic contexts in VSNs. Thus, in this section, we present the key design and corresponding implementation strategies of S-Aframe, and discuss how they meet the requirements of VSN applications.

A. OVERALL ARCHITECTURE OF S-AFRAME

We adopt a hierarchical architecture for S-Aframe, so as to provide a light-weight but scalable framework and facilitate agent collaborations and resource reuse for multiple VSN applications on mobile devices. As shown in Figure 2, the architecture of S-Afame has four layers from the bottom up: the framework services layer, resident agent layer, mobile agent layer, and owner application layer.

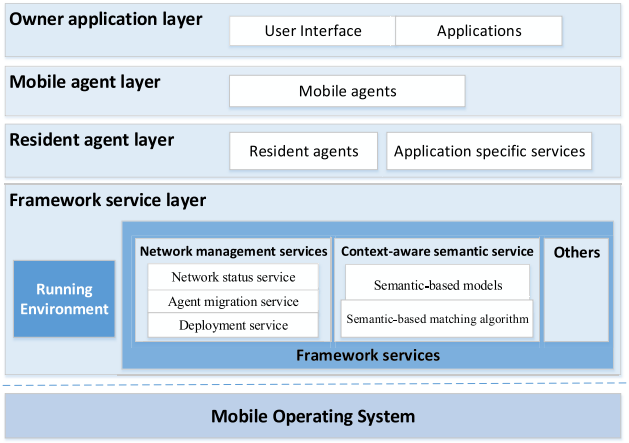


FIGURE 2. **Architecture of S-Aframe.**

1) ARCHITECTURE LAYERS

a: FRAMEWORK SERVICE LAYER

This layer initializes the run-time environments of S-Aframe, extracts the context information from the VSN, and provides the core functions and generic services as framework services to the upper layer. The framework services are only accessed by the resident agent layer but not by other layers. In order to meet the unique requirements of VSN applications beyond basic services like time service and ID name service, this layer consists of two core functions as shown in Figure 2: network management services, and CSS. They enable applications of VSNs to become context-aware in adapting to dynamic network status and dynamic users' contexts in VSNs simultaneously. More details about these two functions will be presented in Section 4.2 and Section 4.3.

b: RESIDENT AGENT LAYER

The resident agents provide all application services of S-Aframe on each node to visiting mobile agents. Resident agents can be deployed automatically on-demand to any mobile node that participates in the same VSN. Once resident agents are deployed to a mobile node, they will remain in it to provide various application services simultaneously as long as the node is a part of the VSN. Mobile agents can directly use all the services that the resident agents provide. Application services provided by the resident agent layer can be built on the services provided by the framework service layer. Framework services only implement basic and generic functions and services that are required by most VSNs applications. Application developers of S-Aframe can develop new application services and deploy them in the resident agent layer. Thus, the services provided by this layer contain two parts: (i) framework services provided by the framework

service layer and (ii) application specific services developed by application developers. Framework services could be reused by every VSN application, while application specific services could be reused by a specific type of VSN application on the mobile devices during run-time.

c: MOBILE AGENT LAYER

The mobile agents run on top of resident agents and are used to execute different application services provided by resident agents. They do not contain any application services in their codes. All the services a mobile agent needs come from the resident agents, such as the services for specific applications, migration service, etc. A mobile agent only contains basic information, such as its migration mode, processing scheme for executing results, as well as computation and communication results. Also, mobile agents are used for transferring necessary resources or application services to some mobile devices when they do not contain such resources or services. Moreover, the data of task execution result gathered from a mobile agent could be shared among multiple VSN applications in one device.

d: OWNER APPLICATION LAYER

The applications are owners of mobile agents. An application provides the user interface to its users who use mobile devices in their vehicles. Through the user interface, the user can select the developed functions he/she prefers, and the owner application of S-Aframe could automatically initiate a mobile agent to execute the services of S-Aframe to accomplish the corresponding function in the VSN system, or release multiple mobile agents to accomplish different tasks simultaneously. S-Aframe supports multiple mobile agents with multiple owner applications working at the same time. In addition, the owner applications can monitor the status of mobile agents they release with the help of resident agents distributed on each node.

2) ARCHITECTURE IMPLEMENTATION

The overall implementation flow of S-Aframe is shown in Figure 3. The technical details of S-Aframe, the list of Java APIs, as well as the source codes of a prototype are available in [26]. The current version of S-Aframe has been implemented based on the Android system. The execution environment and framework services of S-Aframe are developed by implementing a new layer - mobile S-Aframe AmbientTalk library, on top of the AmbientTalk mobile libraries and class libraries of Android. In this layer, we encapsulate the AmbientTalk virtual machine, and its related basic networking APIs as a new library of Android. On top of this layer, we developed the S-Aframe Java class library, and designed a mechanism which can invoke the method *evalAndPrint(String script, PrintStream output)* in the original class library of AmbientTalk to exactly map the class attributes between these two layers. Based on the APIs that we implemented, we developed the framework services of S-Aframe, such as the network management services

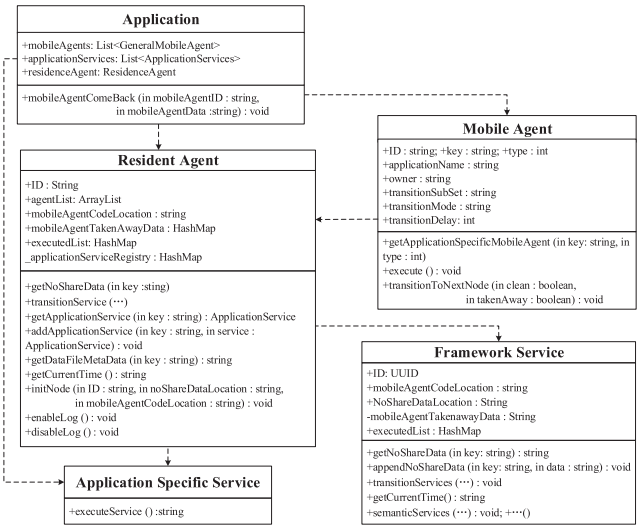


FIGURE 3. Implementation flow of S-Aframe.

and CSS. Thus, the agent-based applications of S-Aframe developed in Java on Android could automatically invoke the APIs in the mobile S-Aframe AmbientTalk library layer.

Then, there are two main implementation tasks of the resident agent layer: (i) configure the framework services as operations and provide them to the upper layers (mobile agent and owner application layers); (ii) configure the application specific services implemented by application developers using Java, and provide these services to the upper layers as well. The mobile agents and owner applications can then access the Java APIs of S-Aframe through accessing the services provided by the resident agent layer. Finally, based on the above, S-Aframe provides the programming model of resident agent, mobile agent and owner application to application developers, who can use Java to program them according to the specification of the programming model and Java APIs.

B. DYNAMIC NETWORK ENVIRONMENT

As mentioned above, we implemented the mobile S-Aframe AmbientTalk library as extended class libraries of Java, which can leverage the advantages of AmbientTalk in our framework, e.g., functions for transparent multi-hop routing in the network layer. Thus, in S-Aframe, we consider two situations when mobile agents are executing the applications while the network environment of VSN changes:

First Situation: Mobile nodes become disconnected when mobile agents are executing applications in a VSN system.

Second Situation: New mobile nodes join a VSN system when mobile agents are executing applications.

In S-Aframe, we develop a novel network status service in its framework service layer, which makes use of the network APIs provided by AmbientTalk, where every node can listen and determine how many nodes are currently available in the VANET. In addition, we adopt a scheme that enables resident agents to share all the IDs of currently connected

nodes, as well as available service lists of mobile devices. By using this service, the framework service layer can inform the upper layer about the real-time networking status of the VSN, such as the current list of node IDs and available services. At the same time, considering the diverse VSN scenarios and specific performance concerns of different VSN applications, S-Aframe does not put any constraint on the application behaviors like specific migration schemes of mobile agents. Instead, to ease the developers' efforts, S-Aframe provides three generic migration strategies for application developers to develop mobile agents: i) migrating among all available mobile nodes; ii) migrating among a subset of mobile nodes; or iii) only migrating to one mobile node (a special case is that the mobile agent does not migrate to any device but executes the application tasks on the local device). Thus, application developers can flexibly select a migration mode, define a migration list and identify primary service requirements (e.g., a specific group of potential drivers who meet the basis requirements in a rideshare application) when they are developing mobile agents to accomplish VSN applications. Moreover, developers can design an algorithm for the migration sequences of mobile agents according to their specific situations.

For the first situation, there are two main issues: (a) disconnection of the target mobile nodes to which mobile agents are migrating; and (b) disconnection of the mobile nodes that are currently hosting some visiting mobile agents. In S-Aframe, a mobile agent can obtain the latest ID name list of currently connected nodes from the resident agent when it migrates to a new node. The mobile agent can also store the list of nodes that it has visited before. Thus, the mobile agent can compare its migration list with the latest ID names list and history list every time it migrates to a new node, in order to automatically decide the next appropriate migrating target. In addition, because the migration time of a mobile agent from one node to another is relatively short (normally less than 3s), the probability that the next visiting target node will become disconnected during a mobile agent's migration is correspondingly low.

Moreover, if a node currently visited by a mobile agent becomes disconnected, it can automatically continue executing the task once the host node is reconnected since in S-Aframe, a mobile agent can store its task history and execution results. Also, the mobile agent's owner application will wait for some predefined amount of time, after which a new mobile agent will be released to continue execution of the application. Consequently, when nodes become disconnected, mobile agents in a VSN system can self-adapt as they are executing their applications upon VANETs. Also, unlike former solutions for agent-based applications on mobile networks, which specified the behaviors of mobile agents [27], or completely ignored the agent migration patterns [12], S-Aframe not only can handle nodes failures in VANETs for different VSN applications, but also enable developers to flexibly develop different customized VSN applications.

On the other hand, network connectivity will change and new service requirements will arise when new nodes join and leave the VSN system. As mentioned above, mobile agents can obtain the latest network connectivity status through resident agents when they migrate to a new node and automatically decide the destinations. Therefore, it will not impact mobile agents' migration when new nodes join and leave the VSN system and when network connectivity changes. The main issue for the second situation is the new service requirements. We assume that all nodes have the initial executing environment of S-Aframe when they join the VSN system. Also, we design a new deployment service in S-Aframe, whereby when a new node joins the VSN system, it will automatically broadcast a request to other nodes. Once an existing mobile node of the system receives the message, it can automatically deploy the framework service and resident agent to the new node. Based on the framework service, the new node can obtain the up-to-date status of the VSN system, such as the latest ID name list of the users and latest list of available services as previously mentioned. Therefore, when a mobile agent migrates to a new mobile node that has just joined the VSN system and does not contain the necessary services or resources for the mobile agent's task, the mobile agent can automatically transfer the services and resources from other nodes to it through the local resident agent, and thus help resident agents to extend application services. Meanwhile, a user of the new node can release a mobile agent to collect the necessary services and resources that it requires. Thus, different from existing mobile agent platforms such as JADE [28] and SPRINGS [18], which adopt a centralized component to track and update the services whenever mobile agents migrate to new nodes, the S-Aframe approach does not even need to be aware of every single agent involved in the VANETs, because in S-Aframe mobile agents can collaborate with resident agents in every node to self-update the services during their migrations.

C. CONTEXT-AWARE SEMANTIC SERVICE

As shown in Figure 2, the design of CSS mainly consists of two parts: semantic-based models, and semantic-based matching algorithm.

1) SEMANTIC-BASED MODELS

In CSS, we mainly consider three types of semantic-based models for VSN applications developed on S-Aframe: (i) application specific service; (ii) context information; and (iii) user-specified information.

a: APPLICATION SPECIFIC SERVICE OF S-AFRAME AND ITS SEMANTICS

An application specific service of S-Aframe is defined as a process unit encapsulating certain business logic and functions, which elucidates what this service does, how it achieves what it does, and how it can be invoked. The traditional service in a service-oriented architecture consists of several information sections: description information, inter-

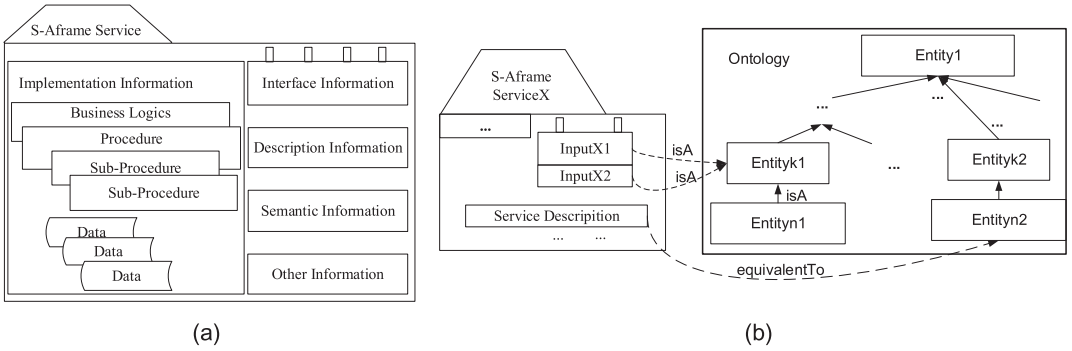


FIGURE 4. Semantic-based model of application specific service of S-Aframe. (a) Application specific service. (b) Modeling semantics of an application specific service.

face information and implementation information. The description information section includes the name, service function, category, and properties of the service. The interface information section specifies a service function via inputs and outputs, and the implementation information contains technical specifications that realize the service function. Each input or output is modeled as a pair of name and data type. For example, a *findPath* service with an input parameter *"to"* with data type *"String"* is represented as *"to:String"*. Although the name of the input parameter may imply that it is a destination address, such semantic information cannot be understood automatically by computers or software agents. In S-Aframe, we follow such a standard service model and adopt these key categories to define all types of information available in the application specific services of S-Aframe, so as to standardize its service interactions. However, we also explicitly incorporate a semantic information section to pave the way for semantics understanding and semantic-based service matching in VSNs. An application specific service of S-Aframe is illustrated in Figure 4(a).

The semantic information section adopts an ontology-based conceptual graph to model semantic information of an application specific service with labeled links. An ontology is a specification of a conceptualization that describes the concepts and relationships existing in a domain. A labeled link connects an item in a service and a concept in the ontology with labels *equivalentTo* or *isA* to model that the item is equivalent to the concept or is an instance of the concept, respectively. Semantics of service functions, service properties, name, inputs, and other information can be annotated in this way. The modeling of service semantics is illustrated in Figure 4(b). For instance, modeling the semantics of the findPath service mentioned previously includes specifying the input "to" as *equivalentTo* the concept Destination. Another example, *535 Robson Street isA Address*, means that the item *"535 Robson Street"* is an instance of the concept *Address*. Note that in S-Aframe, elements in a service may be associated with different ontologies and on the other hand, different application specific services may share the same (i.e., common) ontology.

b: CONTEXT INFORMATION AND THEIR SEMANTICS

There exist various kinds of context information originating from multiple information sources at each node on the VSNs, such as social contacts, user history, vehicular onboard diagnostic (OBD) data and so on. It is hard to enumerate all the context information completely, but it is feasible to classify the core context into several main categories based on the key elements in a VSN: vehicle (Cv), environment (Ce), people (Cp), mobile device (Cm) and network (Cn). Thus, the core context information C can be defined as a set $C = \{Cv, Ce, Cp, Cm, Cn\}$, which can be constructed in the shape of an information tree, as shown in Figure 5(a). Different from the notion of context in [29], which refers to the different meanings of a word in different situations, the context information herein is regarded as a node profile, such as a user profile of a mobile device in VSNs. Its semantics are also modeled with one or more ontology-based conceptual graphs. Figure 5(b) shows an example of context information in two different nodes linked to a common ontology, from which we can infer that vehicle1 is also a car based on the subclass relationship between sedan and car. Application developers can also follow these specifications to define various new application specific services and contexts on top of S-Aframe.

c: MODELING DOMAIN LOGICS AND USER-SPECIFIED PREFERENCES IN LOGIC RULES

Given access to context information, CSS models user preferences using logic rules on how to interpret existing context. A user-specified preference in CSS is modeled as a set of conditions and consequences, which specifies that when the conditions are met (e.g., facts like *"there is a traffic jam on the street ahead"*), its consequences should be taken into consideration (e.g., *"do not drive on the street"*). Each condition and consequence is modeled as a box consisting of a predicate (e.g., *hasTrafficJam*) and one or more variables (e.g., X). Each rule has a link labeled *implies* between the grouped conditions and the consequence; e.g., the rule "*if street(X)* and *hasTrafficJam(X)*, then *notDriveOn(X)*". Such rules are used by CSS with a rule-based inference engine to support the

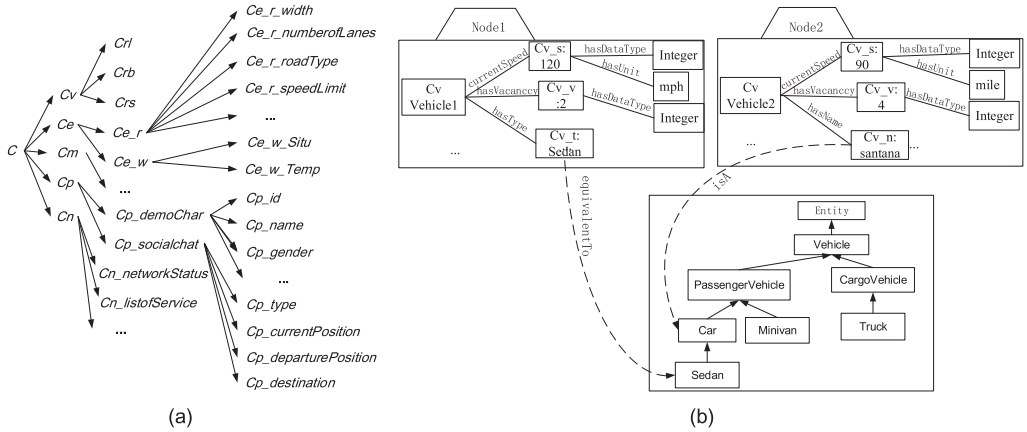


FIGURE 5. Semantic-based model of context information. (a) Context information model. (b) Context information with semantics.

customization of services to individual user profiles and even infer new contexts. CSS also allows application developers or domain experts to represent domain logics in logic rules. For instance, in a carpooling service, the logic "if the number of available seats in a car is greater than or equal to one, then the car is available for carpooling" can be expressed by the rule *"if isCar(X), hasVacancy(X,Y), greaterthan(Y,0), then avaibleCarpooling(X)"*

2) SEMANTIC-BASED MATCHING ALGORITHM

Semantic-based matching of CSS is realized through determination of the extent of mutual satisfaction with respect to the interpreted user request and major functional information of the service, i.e., service name, service functions, service inputs and outputs, and service properties. Semantic matching between a user request r and a service s_i is calculated as the weighted average similarity:

$$Css\,(r, s_i) = \frac{sim_{Css}(r, s_i)}{W_n + W_f + N_1 * W_i + N_2 * W_o + N_3 * W_p} \quad (1)$$

$$\begin{aligned} sim_{Css}\,(r, s_i) &= Sim\left(r, s_{i_{name}}\right) W_n + Sim\left(r, s_{i_{function}}\right) W_f \\ &\quad + \sum_{j=1}^{N_1} Sim\left(r, s_{i_{input,j}}\right) W_i \\ &\quad + \sum_{k=1}^{N2} Sim\left(r, s_{i_{output,k}}\right) W_o \\ &\quad + \sum_{l=1}^{N3} Sim\left(r, s_{i_{property,l}}\right) W_p \end{aligned} \quad (2)$$

$$N_1 = min(N_{r_{input}}, N_{s_{i_{input}}}) \quad (3)$$

$$N_2 = min(N_{r_{output}}, N_{s_{i_{output}}}) \quad (4)$$

$$N_3 = min(N_{r_{property}}, N_{s_{i_{property}}}) \quad (5)$$

where $Sim\,(r, s_i)$ is the semantic similarity between r and s_i, w_n, w_f, w_i, w_o, and w_p are the weights for the name, function, input, output, and property, respectively. $N_{s_{i_{input}}}$, $N_{s_{i_{output}}}$, and $N_{s_{i_{property}}}$ are the number of inputs, outputs, and properties in service s_i, respectively, and $N_{r_{input}}$, $N_{r_{output}}$, $N_{r_{property}}$ are the numbers of inputs, outputs, and properties in r, respectively. In general, we expect that the service name and the service function match well with the user request before we check whether the inputs, outputs and properties match. Thus, two more constraints may be associated with equation (1):

$$Sim\left(r, s_{i_{name}}\right) > \lambda \text{ and } Sim\left(r, s_{i_{function}}\right) > \lambda, \lambda \in [0, 1].$$

Given the context model with semantics in CSS, it is easy to find the corresponding concepts for the two items in the common conceptual graph. For example, for a context model that specifies *"santana isA Car"*, we find the corresponding concept *"Car"* for the term "*santana*". Similarly, we find the corresponding concept for the semantic label "*equivalentTo*". Therefore, computing the semantic similarity between an item in the user and an item in the service can be projected to computing semantic similarity of two concepts.

In a conceptual graph of the common ontology, three aspects impact the semantic similarity of two concepts (elements of the graph): the distance or length of the path between the two elements, the depths of the two elements and the depth of their most specific common parent in the common ontology, and whether the direction of the path between the two elements is changed. The semantic similarity is calculated as:

$$\begin{aligned} &Sim\,(I_1, I_2) \\ &\quad = 2C^{(k+p+d)} depth_{common\_ancestor} / (depth_{I_1} + depth_{I_2}) \end{aligned} \quad (6)$$

where *common_ancestor* is the most specific common parent item; *depth* represents the depth of an item in the ontology; k defines the length difference of the two paths between the most specific common parent item and the two items; p defines the path length between two items; d defines the changes of the directions over the path; and c is a constant between 0 and 1.

a: SEMANTIC MATCHING MECHANISM IN S-AFRAME

In S-Aframe, the resident agent encapsulates the CSS service and provides it to mobile agents and owner applications in the upper layers. New application specific services developed by application developers can be built on top of policies specified by the CSS and communicate with it through the resident agent. When a new mobile device joins the VSN as a new node, it accepts its user's service requirements through the user interface of the owner application, and releases a new mobile agent carrying the user requirements and context information. The new mobile agent then navigates the network, calls the CSS through the resident agent on the mobile node it visits, and carries execution results back to its owner. The semantic matching mechanism of CSS in S-Aframe is shown in Figure 6, which consists of the semantic interpretation procedure, semantic matching procedure, context monitoring procedure, context interpretation procedure, and rule inference engine.

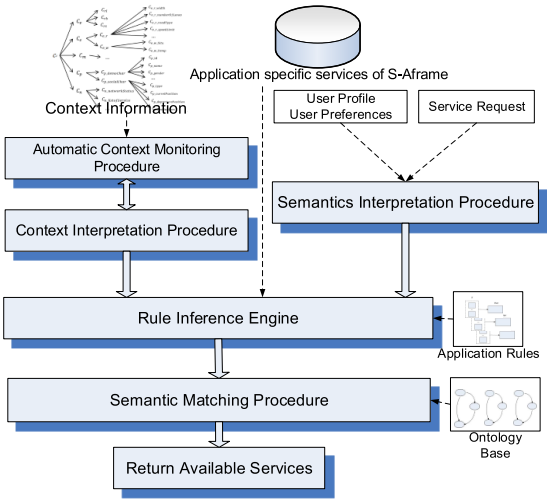


FIGURE 6. CSS mechanism.

Upon receiving a user requirement from the resident agent, the semantic interpretation procedure interprets the user requirement into a semantic conceptual graph if it is a service request, or interprets it into one or more logic rules if it is a user-specified preference. The context monitoring procedure monitors the changes of context in a given VSN through the resident agent and invokes the context interpretation engine to interpret the context information into a semantic conceptual graph upon detecting changes. The rule inference engine is invoked by the context interpretation procedure and the semantic interpretation procedure, takes the detected context information, user-specified rules, a list of available application services and application rules associated with the services available, and performs rule inference. Only services that match all the logic rules can pass the rule inference engine and become available service candidates for the semantic matching procedure. The semantic matching procedure takes the semantic conceptual graph representing a service request, the semantic conceptual graph representing existing context information, and a list of available application services as well as their associated ontologies, and performs the semantic matching function. The semantic matching procedure then returns one or more application services, each of which is associated with a matching degree in line with the predefined threshold, representing the similarity between the service and the user request. The result is handed to the resident agent, and the mobile agent in turn carries the results back to its owner. CSS in the owner node further chooses the service with the highest similarity matching degree and may automatically execute the service to fulfill the user request.

Compared to other existing service systems over dynamic networks [12], [14], [15], the proposed CSS has the following advantages. First, in terms of the range of context information, CSS proposes a comprehensive context model covering the major parties in a VSN. Second, compared to the semantic representation model in [14], CSS adopts a much more straight-forward formula, i.e., labeled links, to establish relationships between services and their semantics, as well as between a context and its semantics, which is more lightweight and scalable for resource-constrained mobile devices. The framework in [15] also applies an ontology-based matching mechanism; however, the similarity measurement may return results contrary to common sense or remain unsolvable in some cases. For example, based on their measurement, the similarity between two concepts Ca and Cb is 0 if they are both direct subclasses of the root concept. Furthermore, to improve interoperability and self-adaptiveness, CSS models not only the semantics of application specific services and context information with labeled links and ontologies, but also the semantics of application domain logics and user-specified preferences information with logic rules. Compared to the framework in [15], CSS also allows application developers and end users to further specify rules on how to interpret context information and model application domain logics. Finally, CSS utilizes a semantic matching mechanism on top of ontology reasoning and rule inference to match user requests and services with respect to specific context information so as to improve its adaptability over mobile dynamic networks.

3) IMPLEMENTATION OF CSS IN S-AFRAME

The current version of CSS in S-Aframe is implemented in Java and built upon several existing technologies including Jena2 [30] and Drools [31]. The Ontology Base consists of ontology-based conceptual graphs that can be easily described in various formats, such as RDF [32] or OWL [33]. Although S-Aframe assumes that domain experts or application developers design and provide such ontologies, they can turn to numerous existing ontologies, such as WordNet [34]. Our current implementation of semantic and context interpretation procedure employs RDF for ontology representation and Jena2 is integrated for parsing RDF. Since CSS is a framework service and does not interact with

application specific services directly, the context monitoring procedure relies on resident agents of VSN applications to monitor the change of context information.

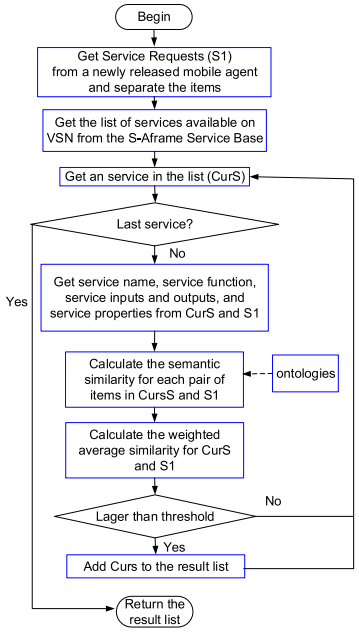


FIGURE 7. Flow chart of the CSS.

Figure 7 shows the flow chart of the semantic matching function in the semantic matching procedure. It is an iterative procedure based on similarity computation as it should go through each item in two sets formed by a service request and application service. The output is a list of services available in the local VSN with corresponding similarity degrees. In addition, in calculating similarity, inference procedures such as retrieving the most specific parent of two items (shown in Algorithm 1), an item's depth in the ontology, the path length between two items, the number of direction changes, and concept projection are implemented. In addition, logic rules in CSS are based on Horn Logic [35] for the inference reasoning. There exist several rule engines for rule inference such as Drools and ALCAS [36]. We integrate Drools, an Object-Oriented Rule Engine for Java in our current implementation.

V. EVALUATIONS OF S-AFRAME

In this section, we evaluate S-Aframe in four aspects that are of particular concerns for VSN users in the real-world: (i) reliability of the developed agent-based mobile applications under dynamically changing network connectivity; (ii) time efficiency of tasks completed by mobile agents across multiple mobile devices in ad-hoc mode; (iii) communication and computation overhead, and memory requirements of S-Aframe on mobile devices; and (iv) effectiveness of semantic-based matching of CSS.

In order for S-Aframe to be evaluated under representative VSN scenarios, the experimental setting is as follows: five users are employed, each carrying an Android device equipped with 802.11n WiFi module. The devices include three Android phones and two Android tablets, all of which are running Android version 4.0 or above and S-Aframe (source code available at [26]). These are common mobile devices that commuters may carry. We use one of the Android devices to act as a WiFi hotspot to which others are connected. As shown in Figure 8, each user carries a mobile device denoted as node *Mx* (where x is the index of each user) and moves in an area of about 150×200 m^2. The inter-node distance ranges from 90m to 250m with an average of approximately 150m. These distances are much longer than the recommended safe distance between vehicles in dense traffic [37], but may be realistic under sparse traffic. There are nine predefined positions, and each user may run S-Aframe's application to initiate mobile agents to accomplish the application tasks from others wirelessly

Algorithm 1 Finding the Most Specific Parent

```
public long FindLeastCommonAncestor(HierarchicalWord
Data[] words, out int distance, out int lcaDepth, out int
depth1, out int depth2)
{
long LCA = -1;
lcaDepth = -1;
depth1 = -1;
depth2 = -1;
distance = int.MaxValue;
int i=-1;
while (++i < 1 && LCA == -1)
{
    IDictionaryEnumerator trackEnum = words[1 - i].
Distance.GetEnumerator();
    if (trackEnum == null) return -1;
    while (trackEnum.MoveNext())
  {
    int commonAncestor = (int)trackEnum.Key;
    if (words[i].Distance.ContainsKey(commonAncestor))
    {
   int dis_1 = words[i].GetDistance (commonAncestor);
   int dis_2 = words[1 - i].GetDistance(commonAncestor);
   int len = dis_1 + dis_2 - 1;
if (distance > len)
{
 int lcaDepth_1 = words[i].GetDepth(commonAncestor);
int lcaDepth_2 = words[1 - i].GetDepth(commonAncestor);
  //lcaDepth = lcaDepth_1 + lcaDepth_2;
  lcaDepth = lcaDepth_1;
  depth1 = dis_1 + lcaDepth_1 - 1;
  depth2 = dis_2 + lcaDepth_2 - 1;
  distance = len;
  LCA = commonAncestor;
...
return LCA;
}
```

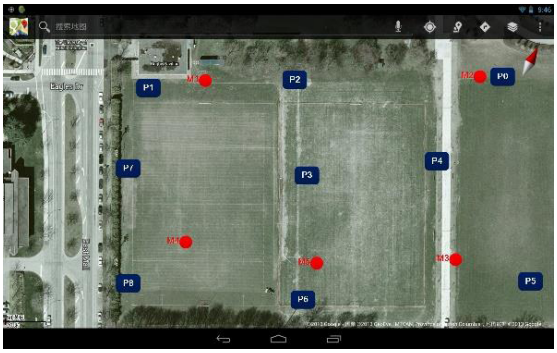


FIGURE 8. **The map area of the experiment.**

when he/she moves from one position to another at normal walking speed. This allows the evaluation of S-Aframe to account for the possible impacts of noise, bad channels, and link failures in real-world environments. Also, we use software configuration methods to let the mobile devices being disconnected/reconnected frequently, so as to simulate the high dynamic feature of VANETs (e.g., vehicles move at high speeds that results in the wireless links of mobile devise on-board vehicles become unreliable and have short lifetimes).

A. RELIABILITY

As discussed in Section 1, the dynamic network connectivity is an inherent challenge for VSN applications. The success rate of task executions in the VSN applications under such dynamism is a suitable measure of reliability.

Due to the experimental constraints, e.g., it is difficult to test the reliability of S-Aframe in multiple vehicles under the same condition, we evaluate the tasks execution success rate of S-Aframe under simulated conditions of mobile devices being disconnected. In a real vehicular scenario, the speed difference between two vehicles travelling in the same direction normally will not exceed 60km/h (16.67m/s), and the sum of the speed of two vehicles in opposite directions normally will not exceed 180km/h (50m/s). Given that the current popular mobile devices equipped with 802.11n WiFi module normally have an ideal WiFi communication range up to 500 meters at a data rate of 15.5 Mbps or higher when there is no obstacle [38], the duration of connectivity between two mobile devices in two different encountering vehicles will be in the range of 10s (500/50) to 30s (500/16.67) or even longer if the vehicles are traveling in the same direction with no excessive speed differential. Therefore, in this experiment, we configure the simulated conditions of mobile devices being disconnected in constant time intervals (T) of 30s, 20s, and 10s, or in exponentially distributed time with rate parameter E = 1/30s, 1/20s, and 1/10s correspondingly. We use two test cases to evaluate this. In the first case, all users initiate a mobile agent from their mobile devices simultaneously to finish the same task (collect the IP addresses of the other four mobile devices). In the second case, all users initiate two mobile agents (one for each task) from their mobile devices simultaneously to finish two different tasks (one is to collect the IP addresses and another is to collect the meta-data of a specific photo from the other four mobile devices). Each case is repeated four times, from which the overall average results are calculated. In addition, during the experiments, we observed that beyond the effect of disconnections by software configuration mentioned above, mobile agents failed to finish their tasks on some occasions due to noise or bad channel conditions, and we also took these cases into account in the results presented.

The results of the first case are shown in Figure 9. We found that S-Aframe achieves similar success rates under a fixed or random time between disconnections. With disconnection in constant time, the success rates are comparable for the cases of T = 30s and T = 20s. In addition, when T = 10s, the success rates degrade faster than those under the two lower disconnection frequencies. With exponentially distributed times between disconnections, the success rates are similar when E = 1/30s and E = 1/20s, and degrade faster when E = 1/10s. As discussed above, since in real vehicular scenarios, the durations of connectivity between two mobile devices in two encountering vehicles are normally more than 10s, the probability that they get disconnected within 10s is

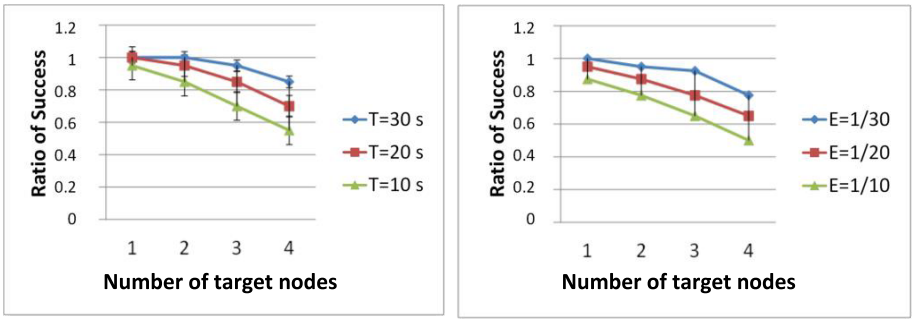


FIGURE 9. Tasks execution success rate-first case. (a) Constant time. (b) Exponentially distributed time.

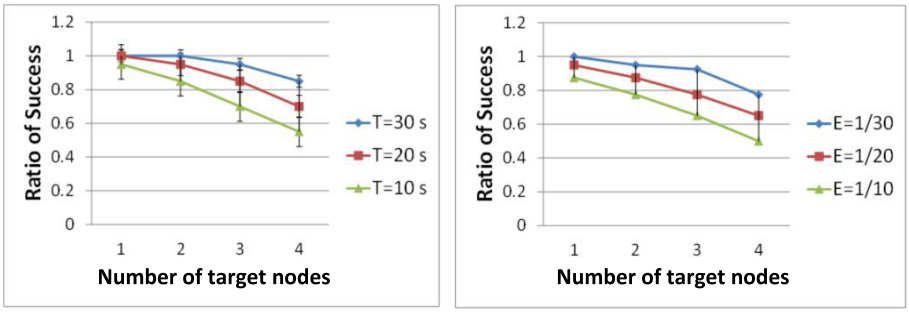


FIGURE 10. Tasks execution success rate-second case. (a) Constant time. (b) Exponentially distributed time.

TABLE 1. Summary of the test scenarios.

Category	Numbers	Objects
Case 1	Owner applications: 1 Types of mobile agents: 1/2 Mobile agents:1/2	To assess the impact of different type of services on task execution latency, and the relative impact between type of services on task execution latency when invoking them concurrently
Case 2	Owner applications: 2 Types of mobile agents: 1 Mobile agents:2	To assess the impact of owner applications on task execution latency when mobile agents invoking the same type of services
Case 3	Owner applications: 2 Types of mobile agents: 2 Mobile agents:2	To assess the impact of owner applications on task execution latency when mobile agents invoking different type of services
Case 4	Owner applications: 2 Types of mobile agents: 2 Mobile agents:4	To further assess the impacts on task execution latency when the above three cases are co-existing

relatively low. Therefore, the applications of S-Aframe can exhibit a high reliability in their task execution in vehicular environments under the considered rates of disconnection frequencies.

The results of the second case are shown in Figure 10. Compared to the first case, under the same disconnection frequencies, the success rates of the second case have degraded a little. It is mainly because in the current version of S-Aframe, its communication is based on AmbientTalk, which is still an experimental language, and its use of multi-cast heartbeats (e.g., UDP packets) to detect disconnected peers may not be very reliable. Thus, the success rates of the S-Aframe tasks may degrade when the communication environments become more complicated. However, the worst success rate in the second case is still more than 50% in the atypical condition when T = 10s or E = 1/10s.

B. TASK EXECUTION LATENCY

Similar to the experimental setting in Section 5.1, we use four test cases to evaluate the execution time efficiency of S-Aframe under the condition of one or multiple owner applications with one or multiple mobile agents in each Android device, which does not go offline in this set of experiments. In the first case, one owner application releases one or two mobile agents to finish the related tasks by using two different types of services: a) original framework services incorporated in S-Aframe; or b) application specific services extended by developers. In the second case, two owner applications release two mobile agents to finish the same tasks by using the same services simultaneously. In the third case, one owner application releases a mobile agent to finish a task by using an application specific service, and another owner application releases a mobile agent to finish a task by using only the framework service. In the fourth case, two owner applications each releases two different mobile agents to finish two different tasks at the same time.

A summary of all test scenarios in this experiment is shown in Table 1. Although S-Aframe supports one mobile agent invoking multiple application services of S-Aframe to accomplish a task, in this experiment, we choose to evaluate under the condition that one owner application in each Android device launches one or two mobile agents, with each agent corresponding to a specific task and finishing the task by invoking one service, so as to better demonstrate the impact of the number of mobile agents and type of application service on the task execution latency of S-Aframe. Moreover, when each owner application has two mobile agents working independently at the same time, we only provide the total time for all mobile agents to finish their tasks in all target nodes, as the task completion times of individual mobile agents are quite close. In addition, we can monitor the status of mobile agents through the user interface of S-Aframe as shown in Figure 11. The results in Figures 12-15 are shown with their 92.5% confidence intervals calculated over the successful rounds (the error margins are difficult to see because they are so small, which are only about ±250.69ms for all cases).

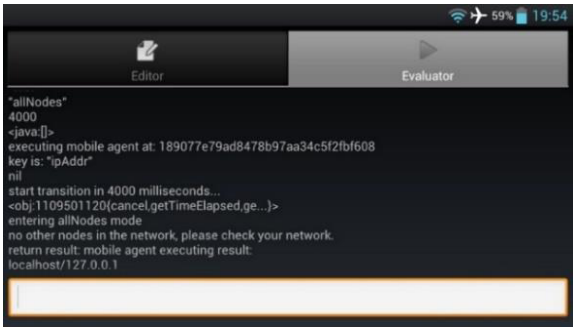


FIGURE 11. Status monitoring of mobile agent.

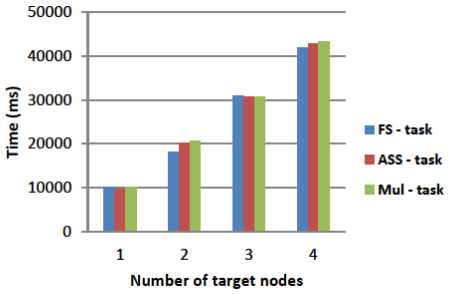


FIGURE 12. Time efficiency of one owner application with one or multiple mobile agents (FS refers to framework service, ASS refers to application specific service, and Mul refers to multiple services).

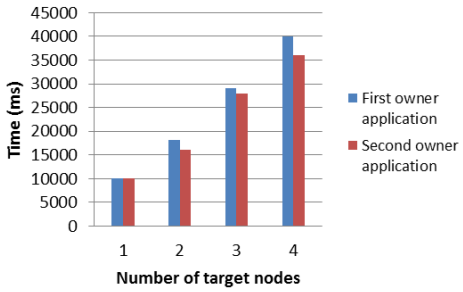


FIGURE 13. Time efficiency (same applications) of multiple owner application.

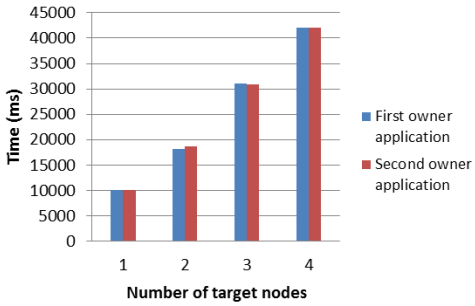


FIGURE 14. Time efficiency (different applications) of multiple owner applications.

From Figure 12, it can be observed that the task execution time latency using the framework service and application

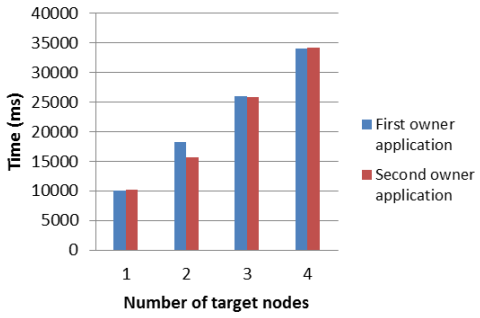


FIGURE 15. Time efficiency of multiple owners with multiple mobile agents.

specific service are comparable. The reason is that after the application specific service is extended by the application developer, it becomes a component of the resident agent layer, and thus mobile agents can directly invoke this service. Meanwhile, as the situations in all mobile nodes are similar, the execution time increases linearly with the number of target nodes. In addition, we find that the execution time of one owner application with two tasks working independently at the same time is similar to that of two tasks working independently at different times. This is because even when two different tasks are working simultaneously, they are in different threads, and thus they do not affect the efficiency of each other.

From the results shown in Figure 13 we find that if two owner applications release multiple mobile agents to execute the same task independently at the same time, the task execution time latency is less than the case when there is only one owner application, in particular when the number of targets increases. Through monitoring the status of mobile agents (as shown in Figure 11), we find that the execution time contains two main parts: migration time (about 2.74s from one mobile device to another) of mobile agents; and processing time (about 4s) for a mobile agent to finish its task in a new mobile device, which consists of initialization time ($<$1s), and the time for processing the task in parallel with fetching the latest information about the network status for migration (about 3s). In the case of multiple owner applications with one or multiple mobile agents, when an earlier agent finishes the task and shares the results with mobile agents that arrive later, time is saved from having to duplicate the tasks. Moreover, from Figure 14, we find that the situation discussed above will not happen if two owner applications release different mobile agents to finish different tasks simultaneously. Therefore, in this case, their execution time latency is found to be similar to that in the case of only one owner application, since they cannot reduce the processing time of tasks. On the other hand, as shown in Figure 15, with multiple owner applications having multiple mobile agents working in multiple related tasks simultaneously, the time efficiency of a single task is usually improved. This implies that the task execution latency of S-Aframe should not increase even if the communication environment becomes more complicated. In our practical

experiment, the inter-node distance is about 150m, and the time for a mobile agent to migrate from one node to another is found to be about 2.74s, or at a speed of 197km/h, which is fast enough to support tasks in many VSN application scenarios, including emergency tasks in traffic accident scenarios.

In addition, as with Agilla [12], we adopted the micro-benchmarks and Java virtual machine (JVM) to perform on a network with 25 nodes, to test the migration of S-Aframe under simulations. We observe that the average one-hop latency, averaged over 100 simulations, is about 146ms, which is lower than that of Agilla (about 210ms) under the same setting. Even though S-Afame and Agilla are implemented on different platforms, a key reason for the better latency performance of S-Aframe is that it uses resident agents to communicate with mobile agents, unlike Agilla which relies on remote shared memory (tuple space) access. This can simplify the operation of the mobile agent on each node and hence decrease the time latency.

C. OVERHEAD

As the communication overhead of each Android device is similar under the same condition, we compute only the average communication overhead (by invoking the network APIs provided by Android) under the condition that each Android device initiates one mobile agent, while other conditions are the same as the experimental setting in Section 5.2. As shown in Figure 16, the communication overhead consists of two parts: foreground and background, where the latter is mainly due to the resident agent and related networking services of S-Aframe and the former is due to the mobile agent when it is visiting different devices in the underlying VANET. As discussed in Section 5.1, the resident agents need to transmit multicast heartbeats at regular intervals to detect any disconnection from peer nodes, thus giving rise to uniform background traffic. This is supported by our observation that the background overhead increases linearly with the number of online nodes, each transmitting 20KB every 5 minutes. Also, since the uniform background traffic is due to the design of S-Aframe and independent of the application, no matter which kind of VSN applications are developed on S-Aframe, the background traffic characteristics should remain similar. On the other hand, the foreground overhead depends on the migration rate (or number of nodes visited per unit time) of a mobile agent. Also, we can foresee that in real-world scenarios of VSNs, since the execution times of mobile agents depend on different purposes of specific applications, which usually follow the exponential distribution, the migration rates of mobile agents also follow the same distribution accordingly. Thus, the foreground traffic of S-Afame can be modeled by a Poisson process. Approximately 12KB is incurred for every move between two nodes for one mobile agent.

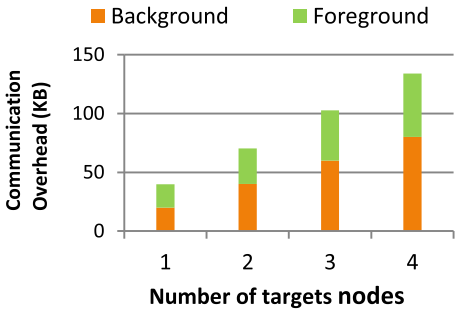


FIGURE 16. **Communication overhead of S-Aframe.**

Also, since the framework services of S-Aframe are implemented using AmbientTalk, we can observe and compare between the communication overheads of AmbientTalk and S-Aframe in the same experimental setting. We find that the background overhead of AmbientTalk is comparable to that of S-Aframe, since both need to transmit multicast heartbeats. On the other hand, their foreground overheads are different as they depend on the implementation methods of VSN applications. For example, since a mobile agent is composed by mobile code, and mobile agents are developed using the Java language in S-Aframe and the AmbientTalk language in AmbientTalk, mobile agents have different code sizes for the two approaches. We observe that using the same migration strategy under the same task, the foreground overhead of S-Aframe is similar to or even lower than that of AmbientTalk. This is because the Android platform natively integrates various Java services and APIs, which may simplify the implementation of mobile agents in S-Aframe by using Java, and hence decrease the code sizes of mobile agents and the related foreground overheads. In addition, as the communication overheads were measured from actual wireless transmissions, the overhead results include the overhead due to transmission failures and retransmissions.

Moreover, we monitored the battery consumption and memory requirement of an Android device (Google Nexus 10 with 9000mAh battery, 2GB memory capacity) running S-Aframe in two scenarios: one in which the Android device with S-Aframe provides application services to visiting mobile agents but do not release any mobile agent; another in which the Android device with S-Aframe constantly releases mobile agents to execute the application tasks. Each of the two test scenarios lasted 30 minutes, and was repeated 10 times. It was found that the average battery consumption and memory requirements due to running S-Aframe were relatively low, consuming only 2% of the battery capacity and 17.2MB of memory in the first scenario, and 2.6% of battery capacity and 19.6MB memory in the second scenario. Also, we compare the battery consumption and memory requirements between S-Aframe and AmbientTalk in the first scenario. We observe that the battery consumption due to running AmbienTalk is comparable to that with S-Aframe ($\sim$2%), while its memory requirement is 15.6MB or $\sim$10% lower than S-Aframe. This is mainly because S-Aframe needs to run more basic services and multiple resident agents while AmbientTalk does not. However, considering that most mobile devices have 1GB or more memory

and are recharged daily by their users at home, workplace, or in vehicle, the battery consumption and memory requirement should not limit the adoption of S-Aframe in most VSN application scenarios.

Furthermore, as the set of performance results from the experiments presented in this section was limited to five users in a specific environment, these results by no means provide a thorough validation of S-Aframe under all possible conditions, and therefore performance comparisons with other situations should be made with caution. For example, increasing the number and density of users, obstacles and weather conditions in other VSN scenarios may also significantly impact the reliability and task execution latency of S-Aframe. The main intention here is to investigate the feasibility of S-Aframe under simple but common scenarios, from which better designs can be realized and applied to different VSNs scenarios for real-world deployment in the future. For instance, we found that from the background traffic studies, while increasing the frequency of multicast heartbeats to detect disconnected peers may increase the task reliability of S-Aframe, it may also significantly increase the cost (i.e., network overhead and tasks execution latency) of VSN applications. Thus, we could investigate a scheme to estimate the disconnection frequency of VANETs, and consider it jointly with the specific application requirements (i.e., high requirements for reliability or not) for dynamic switching of the frequency of multicast heartbeats in mobile devices during run-time, so as to achieve an appropriate trade-off between the task reliability of S-Aframe and the cost of VSN applications in different VSN scenarios. On the other hand, the foreground traffic characteristics of S-Aframe may depend on the type of deployed applications. For applications involving real-time mobile group communications, the foreground traffic generated may exhibit characteristics similar to mobile instant messaging (MIM) [39] that follows an exponential distributions as suggested by 3GPP [40]. However, a more recent study involving an analysis of MIM traffic from 7 million users in a large-scale cellular network suggests that a lognormal distribution may be a better fit for the inter-arrival time distribution of MIM user messages [41]. Thus, further investigation of S-Aframe can be done on the impact of alternative traffic models for the foreground traffic, such as the Joint ON/OFF model proposed in [41]. In addition, in this paper, the underlying network connectivity of S-Aframe is based on pure VANETs, which are highly dynamic networks that may not be always available. If network connectivity is required but a VANET is unavailable, one may need to switch to a cellular network or a nearby WiFi access point as a fallback. In this case, a vertical handover scheme could be designed in the future, to enable S-Aframe to dynamically switch between VANETs and cellular/WiFi networks.

D. EFFECTIVENESS OF CSS

Due to the experimental constraints, it is difficult to find a large number of users and devices for a real-world evaluation of the semantic matching performance of CSS. Thus, we instead evaluate the results of simulating VSNs containing different numbers of nodes ranging from 50 to 1000, distributed randomly over an area of 150×200 m^2, in which each node moves at a predetermined speed and can automatically broadcast a request to other nodes in the network. The faster the nodes move, the more dynamic the network is. Each node in the network provides a list of application services (e.g., returning the information of its vehicle), and has its own context information such as its vehicle information and current position that is updated from time to time. All vehicle information in the context for each node is automatically generated by exemplifying from the Onto_Vehicle ontology. Each node will execute the semantic-based similarity measure matching through CSS, independently from other nodes, when it receives the request broadcasted by the source node. A mobile agent is then released by the source node to access the network. Once the mobile agent reaches a node in the network, it collects the query results.

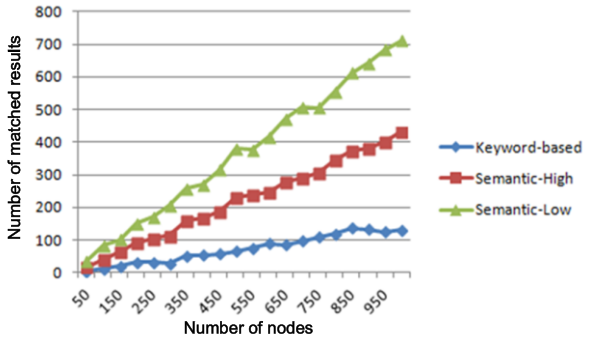


FIGURE 17. **Effectiveness of the semantic-based matching.**

To evaluate the effectiveness of the semantic-based similarity measure, we compare two different matching approaches in the process of finding nodes with similar vehicles: our semantic-based similarity measure matching and keyword matching. In the latter approach, when a mobile agent reaches a node, it recalls the service on the node to return the vehicle information in its context, compares the information with the exact word(s) in the service request that it carries, and determines whether the two have common words or are even identical. It does not take into account the specific meaning of the request and the information from the node. In our semantic-based similarity measure matching approach, the semantic meaning of the vehicle information is acquired by projecting the information onto the ontology, and the similarity between vehicle information of two nodes is returned by recalling the CSS service. We set 0.95 as the value of parameter c in Equation (6) and chose two similarity thresholds: 0.75 and 0.5, for the evaluation.

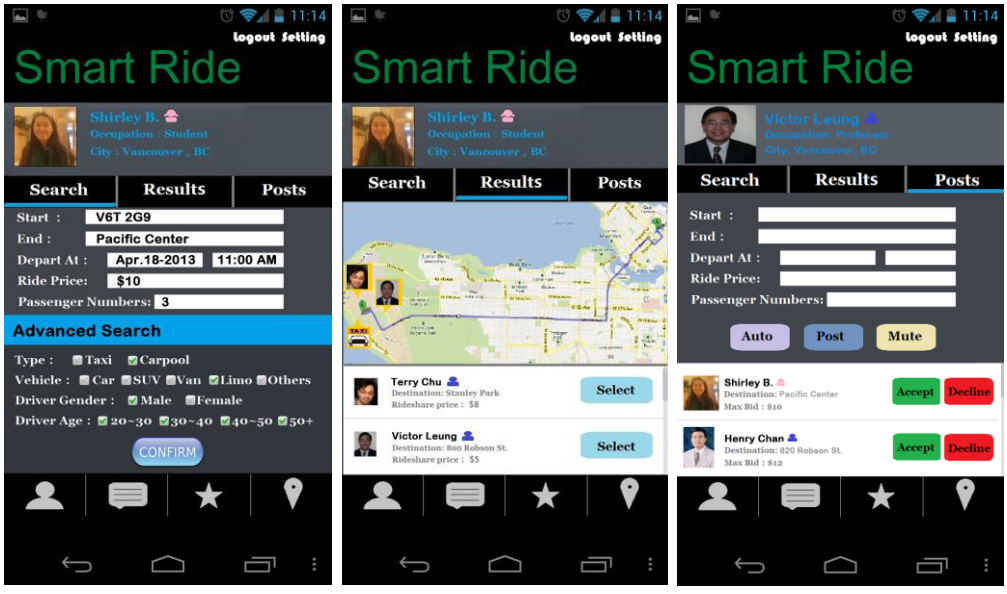
As shown in Figure 17, the results confirm that our approach with a Semantic-High threshold (=0.75) returns more matched results than keyword-based approach as adopted by schemes such as Agilla [12]. Also, the semantic-based matching algorithm in our CSS takes into account

the meaning of the request and measures the similarity on the semantic level. As a consequence, even if the service requested is quite different from that provided on the syntax level, they may still be closely related semantically. This important feature can be applied to realize intelligent service discovery in various VSN applications. Figure 17 also shows that there are more results with the Semantic-Low threshold (=0.5), compared to the higher threshold. That is, with other settings being the same, the lower the threshold is, the more results CSS returns. When the priority in some application scenarios of VSNs is to find a match but the current threshold returns none, such a feature would be important as it guides us to lower the criteria to get more returned results.

VI. APPLICATION EXAMPLE

We present *smart ride*, a novel application based on S-Aframe to meet the requirements for rideshare as discussed at the beginning of this paper. For this application, we first developed a resident agent to provide all the necessary application services at a node to a visiting mobile agent. The resident agent invokes the CSS, which semantically matches the requirements carried by a mobile agent released from a service requestor with the locally available services. If suitable services are found, the mobile agent can return the correct context information and execution results to its owner. Also, beyond generic services provides by S-Aframe, some services like position and map services were developed and deployed as application specific services. Second, we developed the mobile agent and its owner application for this application. The task of the mobile agent is to automatically collect the context information according to the requirements of the users in the underlying VANET by sequentially visiting each one of them. The application creates and sends mobile agents to execute the task in the underlying VANET while provides an interface to the user.

The user interfaces of smart ride are shown in Figure 18. For demonstration, two drivers (driving 2005 Toyota Sienna vehicles) and two potential passengers, each carrying a Google Nexus 4 in which the smart ride application is installed, were deployed to form a VSN. In addition, we observed that the average time to form a VSN among these four nodes was about 8s through a WiFi hotspot. One of the potential passengers named *Shirley* used her Nexus 4 to register with the VSN system and searched for the best-matching driver for ridesharing. Once registered as shown in Figure 18(a), Shirley input her requirements for ridesharing via the user interface. This resulted in S-Aframe automatically releasing a specific mobile agent to execute this service in the VSN. As mentioned earlier, upon visited by a mobile agent, the resident agent can provide it with the necessary application services and CSS. Thus, the mobile agent automatically and intelligently migrated between these phones, executed its task to discover the appropriate drivers, and returned the results to *Shirley*. The average time that Shirley obtains the results is about 18s. In this demonstration, the two drivers were driving at a speed of about 30km/h, and the network was relatively stable. Thus the mobile agent could finish the task with a high success rate (it only failed once in ten repetitions), and the time latency of the task was mainly due to mobile agent's migration and running CSS but not network status update. The migration totally took 12s and running CSS required about 5.1s (1.7s on each node which consists of 1s initial time and 0.7s process time). Furthermore, we found that semantic matching used about 53% of the



(a) (b) (c)

FIGURE 18. Screen-shots of the smart ride user interface.

time (0.37s) running semantic matching CSS in each node, and the remaining time on semantic interpretation, context interpretation, and rule inference. Since the time spent on semantic matching includes several parts such as retrieving the most specific parent of two concepts, and a concept's depth in the ontology, thus the time varies with the size of ontology used for computing semantic similarity of a pair of service items. The current experimental results suggest that, in user applications, small ontologies are preferred to huge ontologies.

Finally, Figure 18(b) shows the results displayed on *Shirley's* phone, which reports that two drivers (*Victor* and *Terry*) are found to match the requirements of her request. Figure 18(c) shows the result displayed on *Victor's* Nexus 4. It may automatically accept *Shirley's* request since it has a high degree of matching. Figure 18(c) also shows the option for ridesharing drivers to similarly post information about their trips and rideshare requirements. In addition, the ridesharing driver could select the *"Auto"* mode, which will automatically post his rideshare information in a specific location/time when he is driving. We further illustrate the matching procedure between a service requester (*Shirley*) and a service provider (*Victor*). First, several logic rules are defined for the application, including the two rules given in Section 4.3.

There are several ontologies necessary for this example, including the domain ontology for service function, which provides different descriptions of the service function in the domain of rideshare and their relationships. Then CSS executes semantic matching and infers that the similarity between *Robson Street* and *Robson Street* is 1 and the similarity between *Sedan* and *PassengerVehicle* is 3/64. Furthermore, the distance matching degree between a service requestor and a service provider is a decreasing function of the distance between them, for example, $f(x) = cos(x.\pi/2D)$, where D is a preset parameter defining the maximum distance that the service provider can be considered as a candidate. Table 2 summarizes the similarity score for each matching item and the overall result at *Time 1*, which shows a 0.81 similarity matching between *Shirley*'s requirements and *Victor's* context information, a 0.65 between *Shirley* and *Terry* and thus the service provided by *Victor* is returned. If keyword-based method is used in this example, *Victor* cannot provide rideshare service to *Shirley* as *Victor*'s service descriptions (e.g., destination) do not match *Shirley's* requirements. Our solution suggests two potential service providers, *Victor* and *Terry* and thus achieves a higher recall rate. Moreover, because the participants in the example are moving all the time, at *Time 2*, the mobile agent detects that *Terry* is a better choice than *Victor*. The semantic based method in MobiSN [15] does not consider any restriction specified by the context information and may return a driver whose vehicle has already reached its full capacity as the service provider. Comparatively, our solution offers a higher accuracy as it further filters service providers through inferences with logic rules.

TABLE 2. Semantic matching results.

Matching Items	Weight	Similarity (Shirley, Victor)		Similarity (Shirley, Terry)	
		Time 1	Time 2	Time 1	Time 2
Service Function	0.5	1	1	1	1
VehicleType	0.15	0.05	0.05	0.21	0.21
Gender	0.1	1	1	1	1
Distance		1	1.5	1.9	0.3
Distance Matching	0.15	0.7	0.38	0.08	0.97
Destination	0.1	1	1	0.09	0.09
Overall	1	0.81	0.76	0.65	0.79

VII. CONCLUSION

In this paper, we have presented S-Aframe, an agent based multi-layer framework with CSS that is able to utilize context information and provide a high level software platform for VSN applications. S-Aframe hides the complexity of dealing with changing network connectivity and varying user services requirements in VSNs, by providing a programming model with extensibility support. S-Aframe also provides a rich set of framework services to support application development using Java with the standard API format. Therefore, S-Aframe provides a modeling paradigm which can facilitate the creation and deployment of different VSNs applications. Furthermore, S-Aframe supports dynamic agent collaborations and service matching during run-time of mobile devices. Thus, more autonomous, intelligent and adaptive applications can be developed for the dynamically changing environments of VSNs.

To the best of our knowledge, S-Aframe is the first mobile platform that supports the development and deployment of context-aware VSN applications. We have extensively evaluated S-Aframe through practical experiments on Android devices, which showed that S-Aframe can perform well under dynamically changing connectivity, and achieve good time efficiency with low computation and communication overhead. Furthermore, we have designed and implemented a ridesharing mobile application based on S-Aframe to demonstrate its practical feasibility for real-world VSNs.

REFERENCES

[1] (2011). *National Household Travel Survey*. [Online]. Available: http://nhts.ornl.gov/download.shtml

[2] N. Liu, M. Liu, J. Cao, G. Chen, and W. Lou, "When transportation meets communication: V2P over VANETs," in *Proc. IEEE 30th ICDCS*, Jun. 2010, pp. 567–576.

[3] A. Lue and A. Colorni, "A software tool for commute carpooling: A case study on university students in Milan," *Int. J. Services Sci.*, vol. 2, nos. 3–4, pp. 222–241, 2009.

[4] L. Han, S. Smaldone, P. Shankar, J. Boyce, and L. Iftode, "Ad-hoc voice-based group communication," in *Proc. IEEE PerCom*, May/Apr. 2010, pp. 190–198.

[5] R. Fei, K. Yang, and X. Cheng, "A cooperative social and vehicular network and its dynamic bandwidth allocation algorithms," in *Proc. IEEE INFOCOM WKSHPS*, Apr. 2011, pp. 63–67.

[6] F. Li and Y. Wang, "Routing in vehicular ad hoc networks: A survey," *IEEE Veh. Technol. Mag.*, vol. 2, no. 2, pp. 12–22, Jun. 2011.

[7] E. Hossain *et al.*, "Vehicular telematics over heterogeneous wireless networks: A survey," *Comput. Commun.*, vol. 33, no. 7, pp. 775–793, May 2010.

[8] N. Do and N. Venkatasubramanian, "Rich content sharing in mobile systems using multiple wireless networks," in *Proc. ACM/IFIP/USENIX Middleware*, 2012, p. 2.
[9] B. D. Higgins *et al.*, "Intentional networking: Opportunistic exploitation of mobile network diversity," in *Proc. ACM MobiCom*, 2010, pp. 73–84.
[10] R. Lu, X. Lin, X. Liang, and X. Shen, "A dynamic privacy-preserving key management scheme for location-based services in VANETs," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 1, pp. 127–139, Mar. 2012.
[11] S. Smaldone, L. Han, P. Shankar, and L. Iftode, "RoadSpeak: Enabling voice chat on roadways using vehicular social networks," in *Proc. ACM SocialNets*, 2008, pp. 43–48.
[12] C.-L. Fok, G.-C. Roman, and C. Lu, "Agilla: A mobile agent middleware for self-adaptive wireless sensor networks," *ACM Trans. Auto. Adapt. Syst.*, vol. 4, no. 3, Jul. 2009, Art. ID 16.
[13] C. X. Mavromoustakis and H. D. Karatza, "A gossip-based optimistic replication for efficient delay-sensitive streaming using an interactive middleware support system," *IEEE Syst. J.*, vol. 4, no. 2, pp. 253–261, Jun. 2010.
[14] K. Fujii and T. Suda, "Semantics-based context-aware dynamic service composition," *ACM Trans. Auto. Adapt. Syst.*, vol. 4, no. 2, May 2009, Art. ID 12.
[15] J. Li, H. Wang, and S. U. Khan, "A semantics-based approach to large-scale mobile social networking," *J. Mobile Netw. Appl.*, vol. 17, no. 2, pp. 192–205, 2012.
[16] T. Van Cutsem, S. Mostinckx, E. Gonzalez Boix, J. Dedecker, and W. De Meuter, "AmbientTalk: Object-oriented event-driven programming in mobile ad hoc networks," in *Proc. SCCC*, Nov. 2007, pp. 3–12.
[17] D. Gavalas, G. E. Tsekouras, and C. Anagnostopoulos, "A mobile agent platform for distributed network and systems management," *J. Syst. Softw.*, vol. 82, no. 2, pp. 355–371, Feb. 2009.
[18] S. Ilarri, R. Trillo, and E. Mena, "SPRINGS: A scalable platform for highly mobile agents in distributed computing environments," in *Proc. IEEE WoWMoM*, Jun. 2006, pp. 633–637.
[19] L. Capra and D. Quercia, "Middleware for social computing: A roadmap," *J. Internet Services Appl.*, vol. 3, no. 1, pp. 117–125, 2012.
[20] N. Abbani, M. Jomaa, T. Tarhini, H. Artail, and W. El-Hajj, "Managing social networks in vehicular networks using trust rules," in *Proc. IEEE ISWTA*, Sep. 2011, pp. 168–173.
[21] D. Huang, Z. Zhou, X. Hong, and M. Gerla, "Establishing email-based social network trust for vehicular networks," in *Proc. 7th IEEE CCNC*, Jan. 2010, pp. 1–5.
[22] X. Hong, D. Huang, M. Gerla, and Z. Cao, "SAT: Situation-aware trust architecture for vehicular networks," in *Proc. ACM SIGCOMM Workshop*, 2008, pp. 31–36.
[23] H.-A. Park, J. W. Hong, J. H. Park, J. Zhan, and D. H. Lee, "Combined authentication-based multilevel access control in mobile application for dailylifeservice," *IEEE Trans. Mobile Comput.*, vol. 9, no. 6, pp. 824–837, Jun. 2010.
[24] M.-C. Chuang and J.-F. Lee, "A lightweight mutual authentication mechanism for network mobility in IEEE 802.16e wireless networks," *Comput. Netw.*, vol. 55, no. 16, pp. 3796–3809, Nov. 2011.
[25] A. Wasef, R. Lu, X. Lin, and X. Shen, "Complementing public key infrastructure to secure vehicular ad hoc networks," *IEEE Wireless Commun.*, vol. 17, no. 5, pp. 22–28, Oct. 2010.
[26] X. Hu. (2014). *S-Aframe*. [Online]. Available: http://mobilesoa.appspot.com/
[27] E. P. de Freitas *et al.*, "Analyzing different levels of geographic context awareness in agent ferrying over VANETs," in *Proc. ACM SAC*, 2011, pp. 413–418.
[28] F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi, "Jade—A java agent development framework," in *Multi-Agent Programming* (Multiagent Systems, Artificial Societies, and Simulated Organizations), vol. 15. Berlin, Germany: Springer-Verlag, 2005, pp. 125–147.
[29] X. Li, S. E. Madnick, and H. Zhu, "A context-based approach to reconciling data interpretation conflicts in web services composition," *ACM Trans. Internet Technol. (TOIT)*, vol. 13, no. 1, Nov. 2013, Art. ID 1.
[30] (2013). *Jena2, Apache Software Foundation*. [Online]. Available: http://jena.apache.org/
[31] (2012). *Drools 5.5, Red Hat*. [Online]. Available: http://www.jboss.org/drools/
[32] D. Brickley and R. V. Guha, "RDF vocabulary description language 1.0: RDF schema," Tech. Rep., 2004. [Online]. Available: http://www.w3.org/TR/2003/WD-rdf-schema-20030123/
[33] M. Dean and G. Schreiber, "OWL web ontology language reference," Tech. Rep., 2004. [Online]. Available: http://www.w3.org/TR/owl-ref/
[34] K. Brown, H. Anderson, L. Bauer, M. Berns, G. Hirst, and J. Miller, *Encyclopedia of Language and Linguistics*. Amsterdam, The Netherlands: Elsevier, 2005, pp. 665–670.
[35] J. W. Lloyd, *Foundations of Logic Programming*. New York, NY, USA: Springer-Verlag, 1987.
[36] B. Spencer. (2006). *ALCAS: An ALC Reasoner for CAS*. [Online]. Available: http://www.cs.unb.ca/bspencer/cs6795swt/alcas.prolog
[37] G. Breyer *et al.*, "Safe distance between vehicles," Centre Effective Dispute Resolution, London, U.K., Tech. Rep., 2010.
[38] (2012). *Ruckus Wireless*. [Online]. Available: http://www.wballiance.com/ wba/ wp-content/ uploads/ downloads/ 2012/07/Ruckus-3G-Offload-Principles.pdf
[39] L.-S. Meng, D.-S. Shiu, P.-C. Yeh, K.-C. Chen, and H.-Y. Lo, "Low power consumption solutions for mobile instant messaging," *IEEE Trans. Mobile Comput.*, vol. 11, no. 6, pp. 896–904, Jun. 2012.
[40] 3GPP TR 43.802 Technical Specification Group, "GERAN study on mobile data applications V0.4.0," 3GPP, Tech. Rep., 2013.
[41] X. Zhou, Z. Zhao, R. Li, Y. Zhou, J. Palicot, and H. Zhang, "Understanding the nature of social mobile instant messaging in cellular networks," *IEEE Commun. Lett.*, vol. 18, no. 3, pp. 389–392, Mar. 2014.

XIPING HU joined the University of British Columbia (UBC), Vancouver, BC, Canada, in 2011. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, UBC. He was the Silver Prize Winner in national Olympic competitions in mathematics and physics in China, and a Microsoft Certified Specialist in web applications, .NET framework, and SQL server. He participated as a Key Member in several research projects, like web service security identification with Tsinghua University, Beijing, China, SAVOIR project with the National Research Council of Canada—Institute for Information Technology, Fredericton, NB, Canada, and NSERC DIVA strategy research network with UBC. As first author, his research contributions have been published and presented in around 20 international conferences and journals, such as the IEEE HICSS, the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, the *IEEE Communications Magazine*, and ACM MobiCom. His current research areas are in mobile social networks, mobile computing, and crowdsourced sensing.

JIDI ZHAO (M'11) is currently an Associate Professor with the School of Public Administration, East China Normal University, Shanghai, China. She received the Ph.D. degree in computer science from the University of New Brunswick, Fredericton, NB, Canada, in 2011, and the Ph.D. degree in management science from Shanghai Jiao Tong University, Shanghai, in 2008. Her research interest lies in interdisciplinary fields covering knowledge management, semantic applications, e-Governance, and crisis response. She has authored over 30 publications in international journals and conferences.

BOON-CHONG SEET (M'05–SM'12) received the Ph.D. degree in computer engineering from Nanyang Technological University, Singapore, in 2005. Upon graduation, he was recruited as a Research Fellow under the Singapore—MIT Alliance Program with the School of Computing, National University of Singapore, Singapore. In 2007, he was awarded a visiting scholarship by the Technical University of Madrid, Madrid, Spain, to pursue research under an EU-funded project on multidisciplinary advanced research in user-centric wireless network enabling technologies. Since 2007, he has been with the Department of Electrical and Electronic Engineering, Auckland University of Technology, Auckland, New Zealand, where he is currently a Senior Faculty Member. In 2011, he was a Visiting Faculty Member with the University of British Columbia, Vancouver, BC, Canada. He was a recipient of the Design and Creative Technologies Faculty Award for excellent in research outcomes from 2012 to 2014. His research activities span the fields of mobile and wireless networking, computing, and communications. He is a member of the Association for Computing Machinery.

VICTOR C. M. LEUNG (S'75–M'89–SM'97–F'03) is currently a Professor of Electrical and Computer Engineering and the TELUS Mobility Research Chair with the University of British Columbia (UBC), Vancouver, BC, Canada. His research is in the areas of wireless networks and mobile systems. He has co-authored over 800 technical papers in book chapters, archival journals, and refereed conference proceedings, several of which received best paper awards. He is a fellow of the Royal Society of Canada, the Canadian Academy of Engineering, and the Engineering Institute of Canada. He has served on the Editorial Boards of the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE WIRELESS COMMUNICATIONS LETTERS, and several other journals. He has provided leadership to the Technical Program Committees and Organizing Committees of numerous international conferences. He was a recipient of the APEBC Gold Medal in 1977, the NSERC Postgraduate Scholarships from 1977 to 1981, the UBC Killam Research Prize in 2012, and the IEEE Vancouver Section Centennial Award.

TERRY H. S. CHU received the B.Sc. (Hons.) degree in computing, in 2010, and the M.Phil. degree in computer science from The Hong Kong Polytechnic University, Hong Kong, in 2013. From 2008 to 2009, he was with IBM, Hong Kong, where he was involved in various commercial software projects. He is the Co-Founder of Bravolol, a leading language learning mobile application platform with over 40 million accumulate downloads and 6 million monthly active users. His research interests include language education, mobile social networking, and image forensics. He received six scholarships, such as Simatelex Charitable Foundation Scholarship in 2008 and 2010, the Bank of East Asia Golden Jubilee Scholarship in 2010, the John von Neumann Scholarship in 2009, and the Chiang Chen Overseas Exchange Scholarship in 2007. He was a recipient of seven awards, including the Hong Kong ICT Awards (Best Innovation and Research Bronze Award) in 2011 and the PolyU Micro Fund Awards (Entrepreneurship Stream) in 2012.

HENRY CHAN (M'98) received the B.A. and M.A. degrees from the University of Cambridge, Cambridge, U.K., and the Ph.D. degree from the University of British Columbia, Vancouver, BC, Canada. In 1998, he joined The Hong Kong Polytechnic University, Hong Kong, where he is currently an Associate Professor with the Department of Computing. His research interests include networking/communications, Internet technologies, and electronic commerce. He was the Chair of the IEEE Hong Kong Section (2012) and the IEEE Hong Kong Section Computer Society Chapter (2008-2009).

SUPPLEMENTARY FILE

1 SEMANTIC SIMILARITY COMPUTING

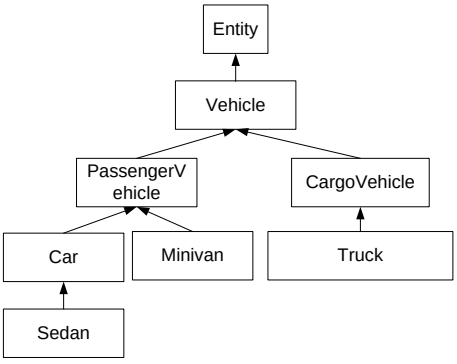


Figure 1. An example of ontology taxonomy

For the semantic similarity computing method presented in Section 4.3-B, here we provide an example to explain Equation (6). Figure 1 shows some part of the taxonomy in the common ontology. First, the distance between two elements refers to how many steps it takes to reach from one element to another in an ontology. We can see that the element *"Car"* goes through four steps to reach the element *"Truck"*. The more steps between two elements, the larger the distance, and the lower the similarity. In this equation, we use the length difference of the two paths between the most specific common parent item and the two items (defined by k) and the path length between two items (defined by p) to define the distance. We can show that $Sim(I_1,I_2)$ decreases when k and p increases as c is less than 1. Second, considering the depth of two elements in the ontology satisfies our intuitive understanding of ontology's classification tree: the classification tree is a progressive refinement work; when tcwo classes are deeper in the classification tree, the relationship between them becomes closer, so that the distance is smaller and the similarity is accordingly higher. For example, the elements *"Car"* and *"Minivan"* are closer than the relationship between the element *"PassengerVehicle"* and *"CargoVehicle"*. Furthermore, two elements have a higher similarity when they are closer to their most specific common parent in the taxonomy tree. Finally, if the direction of the path between two elements changes, the two elements must not belong to the same tree branch and their similarity should be degraded by assigning a lower score. For example, the path direction between the element *"Car"* and *"Minivan"* changes, while the direction of the path between *"Car"* and *"Vehicle"* does not. We can show that the definition of $Sim(I_1,I_2)$ satisfies all these features.

We can see that the range for the semantic similarity is [0, 1]. When two items are irrelevant, there is no common parent, then depthcommon_ancestor is 0 and thus their similarity degree is 0. When two items are identical or synonym or equivalent to each other, we have $k=p=d=0$ and $depthcommon\_ancestor = depthI_1 = depthI_2$, thus the semantic similarity is 1. Let c be 1/2, the similarity degree of *car* and *minivan* is 2*(1/2)(0+2+1)*3/(4+4)=3/32.

2 SEMANTIC MATCHING

For the *Smart Ride* application example presented in Section 6, we reuse the ontology shown in Figure 1 for the vehicle ontology (Onto_Vehicle) and the WordNet ontology as the source of the common onology for items to be matched. Another domain ontology is the road structure ontology (Onto_Road), a fragment of which is shown in Figure 2.

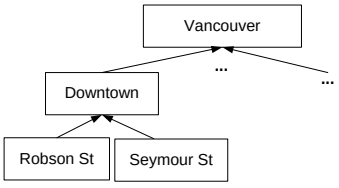


Figure 2. Fragment of the road information ontology

The context information such as the vehicle information, users' personal information, and social information for rideshare is shown in the Figure 25.

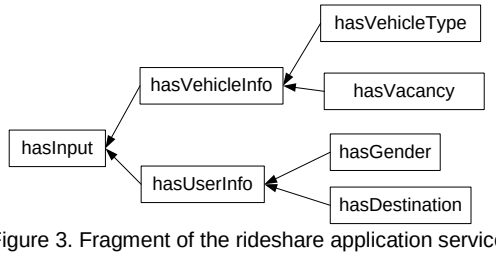


Figure 3. Fragment of the rideshare application service

Next, we exemplify the service request from *Shirley*. Figure 4 shows a fragment of the rideshare requesting service carried by a mobile agent from *Shirley's* mobile phone. CSS infers that *Pacific Center* is an instance of *Robson Street*.

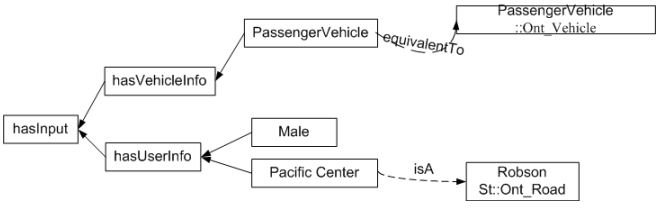


Figure 4. Fragment of the service request

- *Xiping Hu and Victor C. M. Leung are with The University of British Columbia.*
- *Jidi Zhao is with East China Normal University.*
- *Boon-Chong Seet is with Auckland University of Technology.*
- *Terry H. S. Chu is with Bravolol.*
- *Henry C. B. Chan is with The Hong Kong Polytechnic University.*

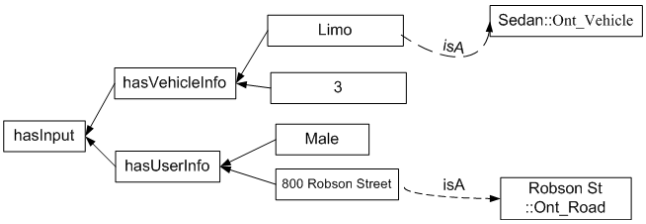


Figure 5. Fragment of the service from Victor

Figure 5 shows a service fragment from service provider *Victor*. The attribute value "3" indicates that *Victor* has three seats available in his *Limo*. Thus CSS derives from the rule inference engine that *Victor* can provide the rideshare service. Furthermore, CSS infers that *Limo* is an instance of *Sedan* and his destination is an instance of *Robson Street*.