**SocioTelematics**: Harnessing social interaction-relationships in developing automotive applications

Muhammad Ashad Kabir*, Jun Han, Alan Colman

Centre for Computing and Engineering Software Systems, School of Software and Electrical Engineering, Swinburne University of Technology, Melbourne, Australia

**ABSTRACT**

In a cooperative convoy, a vehicle interacts with other vehicles, service providers and infrastructure systems to make the travel safe and convenient. Through these interactions a vehicle can share its domain-specific information – acquired from service providers and infrastructure – with other vehicles in the convoy. Such interactions are subject to defined agreements and constraints between the entities (i.e., vehicle to vehicle, vehicle to service provider, and so on), which we refer as (social) interaction-relationships. Such relationships, however, may need to adapt with the changes of requirements. Also a driver may want to automate certain interactions to reduce distraction during driving. A cooperative convoy telematics system should support collaboration (i.e., allow drivers to share specific travel information) and coordination (i.e., allow drivers to automate interactions), and be able to adapt to cope with the changes of requirements.

In this paper, we address these issues and demonstrate how our social interaction-relationships modelling technique can be exploited to develop a telematics system, called SocioTelematics, providing such functionalities. This system allows collaboration and coordination preferences explicitly specified and updated to cope with the changes. In particular, our service oriented implementation enhances adaptability of the system, making it easily deployable and changeable. We have implemented a prototype system based on a client–server architecture where the client application is developed for Android and the server is running on the Amazon cloud. The system's performance and resource consumption were quantified using real life experiments that show the feasibility of our approach.

© 2014 Elsevier B.V. All rights reserved.

**1. Introduction**

Automotive software has increasingly become one of the major enabling technologies for vehicles due to the integration of advanced functionalities, wireless network infrastructures, and novel interactions such as V2X (vehicle-to-any other relevant entity). Telematics systems are emerging automotive technologies that combine advanced communication and vehicle technologies to improve vehicle control and safety, provide comfortable driving, make vehicles environment-friendly, and enhance driver experience [1]. In general, automotive telematics systems (referred to as telematics for short) deal with different types of V2X interactions. In-Vehicle (inV) telematics support seamless interaction between users’ portable devices (e.g., phones, MP3 players, etc.) and built-in vehicle infotainment systems. This allows occupants in a vehicle to be entertained and/or to carry on their work. Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) systems
allow vehicles to “talk” to each other and to roadside infrastructure to share safety warnings, detect hazards and proactively avoid collisions. Vehicle-to-Service Provider (V2SP) systems allow the driver to access value-added services provided by third party providers (e.g., location-based services, travel guide services, etc.).

This range of V2X interactions together with the recent advancement of Internet and mobile technologies illustrates significant potential to develop more intriguing user-centric applications such as “cooperative convoy telematics”—where drivers can share information with each other to make travel safe and convenient. Such V2X interactions, for example vehicle-to-vehicle, are subject to defined agreements and constraints, called social interaction-relationships, between the drivers of those vehicles. These interaction-relationships may need to adapt with changing requirements [2]. Moreover, telematics development calls for the accommodation of competing considerations such as higher demands on safety, interoperability and driver convenience vs. lower production and upgrade costs; longer product life-cycles vs. less effort and shorter development time [3]. It has already been identified that the use of the model-based approach aids in the reduction of product cost and the increase in product quality [4].

Automotive software systems have been rapidly evolving with the proliferation of Internet technologies (such as cloud computing, and service oriented computing) and the ever growing popularization of mobile devices, especially smart phones. DENSO corporation in Japan – a leading supplier of advanced automotive technology – recently proposed an Automotive Cloud Service System software architecture for integrating vehicle software with the cloud services [5]. In [6] authors discuss the way service-oriented computing (SOC) can enhance the adaptability of automotive systems by taking into account the properties of the environment in which they operate. Thus, the recent shift from hardware to software provides new opportunities for telematics systems. Moreover, the scientific community has recently articulated socially-inspired car to car interactions [7].

In this paper, we present a social interaction-relationships based approach to address the requirements of developing a novel telematics system (called SocioTelematics) that supports cooperative convoy. This research has four major contributions:

1. We present our approach to designing and developing SocioTelematics systems starting from analysing and modelling interaction-relationships between a vehicle and other entities.
2. We also present our approach to managing adaptation (i.e., modification) in such interaction-relationship models to cope with the changes in user requirements.
3. We introduce a system architecture together with a development process, a corresponding tool chain for service-oriented software development, and a mobile client application for Android.
4. Finally, we demonstrate the feasibility and applicability of our system with respect to resource consumption and performance using real-life experiments.

The essence of our approach is the externalization of the interaction-relationships from the applications and the explicit modelling of such relationships from domain- and player-perspectives. The domain-centric social interaction model (DSIM) captures a collaborative view of the interaction relationships among the users/actors, whereas the player-centric social interaction model (PSIM) captures an individual’s coordinated view of all its interactions (across different domains), and thus supports coordination. We separate functional and management operations in such runtime models (i.e., DSIMs and PSIMs), which facilitate their adaptation to cope with the continuous changes in users requirements and environments. These runtime models are realized through a platform, and are used to mediate social interactions among drivers in a cooperative convoy. Furthermore, our service-oriented implementation and model-driven approach with tool support make the whole development and deployment process easy.

This paper significantly revises and extends [8] by enhancing it in several directions: (i) providing a detailed requirements analysis; (ii) introducing design goals and features to address the identified requirements; (iii) introducing adaptation management; (iv) elaborating prototype implementation; and (v) reporting adaptation related experimental evaluation.

The paper is organized as follows. Section 2 presents a motivating scenario for a cooperative convoy system, and identifies key requirements for designing and developing such a system. Section 3 introduces design goals and corresponding design features for such systems, while Section 4 discusses our approach to support such features. Section 5 presents interaction-relationships models for a cooperative convoy, while Section 6 introduces runtime adaptation support for such models. Section 7 presents a prototype implementation. Section 8 reports our experimental evaluation. Section 9 reviews related research and discusses the key benefits provided by our approach. Finally, Section 10 concludes the paper and highlights the future work.

2. Motivating scenario and requirements analysis

In this section, we identify the key requirements of developing SocioTelematics systems by analysing a cooperative convoy scenario.

2.1. The cooperative convoy

Two groups of tourists in their two cars hired from two different rental companies want to drive together from Melbourne to Sydney. The car rental companies provide different levels of support to their customers based on the customers’ insurance policies, payment, service availability, and so on.
Let us assume that the two cars, Car#1 and Car#2, as depicted in Fig. 1, are rented from Budget and AVIS respectively. The Budget rental company provides a car with a wet-sensor that can detect relevant road conditions and give an alert to reduce speed as necessary. The company also allows access to a travel guide service (TGS) that suggests a possible route to a destination considering the shortest path. However, the service is unable to report real-time traffic information, e.g., road blocks due to road accidents. In contrast, the AVIS rental company allows its car to access a traffic management service (TMS) that provides information on traffic conditions ahead, e.g., the road is blocked due to an accident or emergency road works. However, both of these car rental companies provide a roadside assistance service (RAS) to their customers. When a vehicle experiences a mechanical issue, its telematics system automatically informs the rental company and RAS, and requests a tow truck if the vehicle is no longer drivable. Information about the vehicle location and the mechanical issue is also sent with the request. Upon receiving the request, the road-side service validates it with the car rental company and takes the necessary steps as defined in the agreement with that company.

Drivers of Car#1 and Car#2 want to form a cooperative convoy to make their travel safe and convenient by collaborating and interacting with each other. In a cooperative convoy, a vehicle interacts with other vehicles, service providers and infrastructure systems to make the travel safe and convenient. Through these interactions a vehicle’s driver can share information (acquired from the service providers and infrastructure systems) with other vehicles’ drivers in performing their tasks. Such interactions are subject to defined agreements and constraints among the entities (i.e., vehicle to vehicle (V2V), vehicle to service providers (V2SP), and vehicle to infrastructure (V2I)).

In the above scenario, we can see that Car#1 and Car#2 have access to different types of services: Car#1 has access to a travel guide service, while Car#2 has access to a real-time traffic management service. In the cooperative convoy, they decide that Car#1 is the leading car (LC) whilst Car#2 is the following car (FC). The two cars follow the same route chosen by the leading car as it has access to a TGS. In addition, the drivers of both cars agree on a number of issues. For instance, they will always keep the distance between them less than 1 km. Car#2 (the following car) will send road blocks information (obtained from its TMS) to Car#1 (the leading car) if there is any. Car#1 will get the updated route plan from its TGS by specifying its preferences (e.g., avoid the route with blocked road) and will notify that route information to Car#2. Both cars will notify each other of their positions every 10 s. If either vehicle experiences a mechanical problem (e.g., flat tyre, engine issue) it will notify the other vehicle.

2.2. Requirements analysis

The telematics application facilitating the cooperative convoy needs to fulfill three major requirements:

First, the application should support interactions complying with the agreed interaction-relationships (i.e., constraints and obligations). In the above scenario, these interaction-relationships can be clustered into three groups by taking a collaborative view:

- **Interaction-relationships between the driver of LC (Car#1) and the Budget Company with its associated services**: The interaction-relationships such as getting route plan from the TGS, and requesting road side assistance in case of any mechanical problem, fall into this category.
- **Interaction-relationships between the driver of FC (Car#2) and the AVIS Company with its associated services**: The interaction-relationships such as getting road blocks information from the TMS, and requesting road side assistance in case of any mechanical problem fall into this category.
- **Interaction-relationship between the drivers of LC and FC**: The interaction-relationship between drivers of LC and FC (i.e., their mutual agreements) falls into this category. It includes the requirement that FC should send information on any road blocks to LC if there is any, LC should notify the updated route information to FC, FC follows the same route chosen
by LC, both cars notify each other of their positions every 10 s, and in case of any mechanical problem both cars notify each other, and so on.

Thus, a model is required to capture the collaborative view of such interaction-relationships.

Second, the application needs to provide a coordinated view of the interactions, and to allow drivers to specify their coordination preferences and to perform the coordination in an automated manner. According to the scenario, each driver is responsible for performing several tasks, for instance, the driver of LC is responsible for notifying route information, current positions, and mechanical problems (if any) to FC. Similarly, the driver of FC should send road block information, current positions, and mechanical problems to LC. Drivers performing such additional tasks (e.g., forwarding information) may cause distraction and have undesirable consequences. Thus, to facilitate collaboration with less distraction, it is required to provide a coordinated view of the interactions and to allow drivers to specify their coordination preferences.

Third, the application needs to support runtime adaptation, as the interaction-relationships evolve over time and thus need to adapt with the changes in requirements and environments. For instance, a third vehicle could join when the convoy is on the way; or the break-down of a following vehicle might result in its leaving the convoy before reaching the destination. A mechanical problem of the leading car may require it to handover the leading car role to one of the following cars. Because of heavy rain, the maximum distance may need to be reduced from 1 km to 600 m.

3. Design goals and features

The above requirements analysis of the SocioTelematics application identifies three main design goals: (1) the application should support social interactions and enforce users' agreements in such interactions, (2) the application should provide a coordinated view of the different interactions and allow its users to perform coordination in an automated manner, and (3) the application needs to support adaptation in interaction-relationships at runtime.

These design goals can be addressed by three design features:

- **Modelling runtime social interactions from a domain perspective**—To address the first design goal, it is required to model and represent users' agreements related to their social interactions. This model should have a runtime representation to enforce constraints and agreements while social interactions are happening. We provide a role based approach to model and represent users' agreements related to their social interactions, namely the domain-centric social interaction model (DSIM) (see Section 5.1). Our middleware platform (see Section 7.1) provides a runtime environment to instantiate the DSIM and to facilitate social interactions complying with the model specification.

- **Modelling runtime social interactions from a player perspective**—To address the second design goal, it is required to model and represent social interactions from a player (user) perspective, to allow the player to specify his/her coordination preferences, and to support execution of these preferences. Our player perspective of modelling social interactions, namely the player-centric social interaction model (PSIM), represents a player’s coordinated view of all his/her social interactions (see Section 5.2). Like for a DSIM, the platform also provides a runtime environment for instantiating a PSIM.

- **Supporting runtime adaptation**—To address the third design goal, i.e., allow one to modify runtime social interaction models, it is required to support runtime adaptation in such social interaction models. It has already been recognized that the model based development approach is key to facilitating the application development process and managing runtime adaptation [9–11,4]. In our approach, inter-actor relationships in an application are (externalized from the application implementation and) modelled explicitly using DSIMs and PSIMs, their runtime (execution) environment is generated, and adaptation is managed by the middleware platform.

4. Our approach

In this section, we present an overview of our approach to modelling and managing social interaction-relationships, which we exploit in developing SocioTelematics.

Role based approaches have been increasingly used in software system design and implementation. Roles, in different variations, have been applied in many fields such as computer supported collaborative works, role based access control, multi-agent systems, and object modelling [12]. In the access control domain, the notion of roles has been used to represent a set of responsibilities and privileges associated with a particular position within an organization [13]. In CSCW and group-aware applications, the notion of role has been used to provide a natural abstraction for representing users without requiring prior knowledge of their individual identities [14,15]. In multi-agent systems, the concept of role has been used as an abstraction for representing an entity's behaviour [16], where a role specifies a functional behaviour of an agent.

We have taken a role based approach to model social interactions as it has been evaluated as being very useful in modelling entities, their relationships, functions and interactions [17]. Our approach utilizes the notion of roles for functional abstractions. Among the many different approaches to role modelling [18], one common way to develop a role based model is to use organizational concepts [19]. According to Daft [20] “organizations are social entities that are goal directed, deliberately structured, and coordinated activity systems, linked to the environment”. A similar view has been taken by Rollinson [21], who says “organizations are artefacts, that are goal directed social entities with structured activities. Organizations are typically viewed as a managed network of roles that decompose the abstract functions of the organization into descriptions that can be performed by role players (e.g., people, subsystems, other organizations). Organization theory addresses the structuring
of organizations in order to more effectively achieve those goals (e.g., [22]). The structure defines how the organization is composed in terms of its constituent units, and the relationships or the connections between these units.

We adopt such a view of organization as a composition structure of dynamic relationships between social roles in order to model runtime social interactions. Role Oriented Adaptive Design (ROAD) [23] introduces such an organizational structure to define self-managed software composites. In this research, we adopt ROAD and further extend it to define adaptable runtime social interaction models. ROAD defines roles or positions of a composite. Players are dynamically bound to the composite to play these roles. Among many different approaches to the design of role-based software systems using agent paradigm (see [17] for a survey), ROAD brings a number of design principle that support flexible management and runtime adaptations, which has also been recognized and appreciated in [24]. One such principle is the separation of functional and management operations. This principle is important for facilitating application development and runtime adaptation as the functional operations performed by the actors enable normal interactions while the management operations realize adaptation to the structure and parameters of the model. Thus, we adopt the ROAD architectural style for modelling and managing interaction-relationships at runtime.

We enrich the interactions in ROAD with the new constructs of obligation and conversation. We specialize the ROAD composite structure (meta model) from the domain and player perspectives [25]. To provide a collaborative view of runtime social interactions among actors, we propose a domain-centric social interaction model. To facilitate collaboration, the applications need to provide a coordinated view of the interactions, allow a user to specify his/her coordination preferences and perform the coordination in an automated manner. To address this requirement, we further propose a player-centric social interaction model. We provide graphical notation and have also developed XML schemas for both the DSIM and PSIM. For the sake of readability, in this paper, we use a template for presenting models by using a pseudo notation on the XML schemas. In this notation, the terms in boldface represent the XML tags in our schema (see Section 5 for examples).

A promising approach to support runtime changes is to develop adaptation mechanisms that leverage software models, referred to as models@run.time [26]. Runtime models provide “abstracts of runtime phenomena” [27]. That is, the models should represent the system and should be linked in such a way that they constantly mirror the system and its current state and behaviour; if the system changes, the representations of the system – the models – should also change, and vice versa. A runtime model can be seen as a live development model that enables dynamic evolution and the realization of software design [26]. Taking this view, we realize DSIMs and PSIMs as runtime models and manage their runtime adaptation through a middleware platform.

5. Modelling social interaction-relationships

5.1. Modelling collaboration using domain-centric models

The Domain-centric Social Interaction Model (DSIM) provides a structure of the interaction-relationships among social roles associated with a particular domain or environment such as a company, a cooperative convoy, and so on. The DSIM groups the social interactions from a domain perspective.

In the above scenario, there are three domains that need to be modelled, namely those concerning the two rental car companies and the cooperative convoy. The two cars have been hired from two different companies and they allow different types of services. For instance, the Budget Rental Company allows travel guide and roadside assistance services, and the company has its own policies and constraints to use those services through the telematics system in the cars rented from them. These constraints and policies can be captured in the domain-centric social interaction model of the Budget, called BudgetDSIM, as shown in Fig. 2(a). This model comprises four social roles (RentedCar, RentalCompany, TravelGuide and RoadsideAssistance), four relationships (R1, R2, R3, and R4), and one organizer role. These relationships capture the interaction constraints (as specified in the scenario) among the social roles. For instance, R1 (RentedCar–TravelGuide) relationship represents that “upon receiving a route request from a RentedCar, TravelGuide should send a route notification by 15 seconds”. Fig. 3 shows the representation of this agreement which consists of two social interactions (i7 and i8), one conversation (c1) and one related obligation (o3). The i7 social interaction represents the routeRequest interaction from RentedCar to TravelGuide, while the i8 represents the routeNotification interaction from TravelGuide to RentedCar. The c1 conversation imposes the sequence related constraints between these two social interactions, while o3 imposes time related obligation.

The Budget company is the owner of this runtime model and controls it through the organizer role. The customer, e.g., the tourist group, who rents Car#1 plays the RentedCar role through the telematics system of that car. The roadside assistance service and the travel guide service play the RoadsideAssistance and TrafficManagement roles respectively.

On the other hand, the AVIS company allows access to traffic management and roadside assistance services. Like Budget, we also model the domain-centric runtime social interactions for AVIS, called AVISDSIM, as shown in Fig. 2(b). It also consists of four social roles (RentedCar, TrafficManagement, RentalCompany and RoadsideAssistance). Unlike Budget, it does not have a TrafficGuide role. However, it has a TrafficManagement role. There are four interaction-oriented social relationships, i.e., R5, R6, R7 and R8, among these roles, which represent agreements and constraints among social roles, according to the specification in the scenario. The AVIS company plays the organizer role of this runtime model as the company is the owner of this model. The customer, e.g., the tourist group, who rents Car#2 plays the RentedCar role through the telematics...
Fig. 2. Domain-centric social interaction models (a) BudgetDSIM, (b) AVISDSIM, and (c) ConvoyDSIM.

system of that car. The roadside assistance service and the traffic management service play the RoadsideAssistance and TrafficManagement roles respectively.

Like Budget and AVIS, the interaction-relationship between the drivers of Car#1 and Car#2 is modelled from a domain perspective, called ConvoyDSIM, as shown in Fig. 2(c). According to their agreements, it consists of two social roles: LeadingCar and FollowingCar, and their interactions are captured in the R1 (LeadingCar–FollowingCar) interaction-relationship. The Car#1 plays the LeadingCar role, while the Car#2 plays the FollowingCar role. Car#1 is also designated to play the organizer role of the ConvoyDSIM. Fig. 4 shows a partial notational description of ConvoyDSIM. It includes description of the i1 social interaction, and its associated obligation (o1) and operational parameter (p1), which represent the agreement “FollowingCar should send a position update (as a positionUpdate interaction) to the LeadingCar every 10 seconds”.

5.2. Modelling coordination using player-centric models

As presented in the previous section, the R9 interaction-relationship of ConvoyDSIM captures the agreements between leading car (Car#1) and following car (Car#2) in terms of their social interactions. But some of these interactions are driven by or depend on their interactions with other associated domains (i.e., Budget and AVIS). For instance, when the road block information is received from the traffic management service, following car should immediately forward it to the leading car. Likewise, when the route information is received from the travel guide service, leading car should forward it to the following car. However, in some cases the driver may want to carry out these activities automatically with less or no explicit human intervention. For example, the driver of following car may want to forward the road block information to the driver of leading car as soon as it is received from the traffic management service. The player-centric model addresses these issues by introducing the Coordinator role which allows the player to coordinate (automatically or manually) his/her social interactions.

Fig. 5 shows the player-centric social interaction models of Car#1 and Car#2, called Car#1PSIM and Car#2PSIM respectively, and how they relate to the domain-centric models (i.e., BudgetDSIM, ConvoyDSIM and AVISDSIM). According
Fig. 3. A partial description of R1 relationship in BudgetDSIM.

to the cooperative convoy scenario, Car#2 plays the FollowingCar role in ConvoyDSIM and RentedCar role in AVISDSIM. Thus, the Car#2PSIM contains FollowingCar and RentedCar social roles and their associated role-centric social relationships, i.e., Rr3 and Rr4. The Rr4 is the role-centric social relationship of RentedCar social role in AVISDSIM. Therefore, Rr4 is the aggregation of R5, R6 and R7 (social) interaction-relationships in the AVISDSIM. The Rr3 and Rr4 connect the Coordinator role to the FollowingCar and RentedCar roles respectively. Similarly, Car#1PSIM consists of LeadingCar and RentedCar social roles and their associated role-centric social relationships, i.e., Rr1 and Rr2, which connect these social roles to the Coordinator role.

A player/actor plays social role(s) in different DSIM through its PSIM. For instance, the Car#1 plays RentedCar role in BudgetDSIM and LeadingCar role in ConvoyDSIM through the Car#1PSIM. Similarly, the Car#2 plays RentedCar role in AVISDSIM and FollowingCar role in ConvoyDSIM through the Car#2PSIM.

In the PSIM, the coordinator (role and player) does the most important work as it performs actions on behalf of the player based on his/her preferences. These preferences or coordination rules are player specific and can be defined using Event-Condition-Action (ECA) rules. For instance, the Car#2 driver’s preference of automatically forwarding road block information, received from traffic management service, can be represented using the following ECA rule:

```
Coordination-Rule "Preference of the Car#2 driver"
when OnNotificationReceived //Event
then if (?receivedNotification == notifyRoadBlock) //Condition
do FollowingCar.notifyRoadBlock //Action
```

6. Managing runtime adaptation

Social interaction models (i.e., PSIMs and DSIMs) externalize users’ interaction-relationships from the applications. However, the requirements of such applications are subject to continuous change. Thus, both the PSIMs and DSIMs need to
be managed and adapted to ensure the proper functioning and evolution of the applications in which the social interaction models play a part [28].

The organizer role provides the management capability of a runtime social interaction model. The principle of separation between a role and its player is also applied to the organizer role. The organizer role is internal to a runtime model and allows its player to manage both the structure and parameters of the runtime model. The organizer player can adapt the structure of the runtime model by adding and deleting social roles and relationships, and by binding and unbinding a player to and from a social role. The organizer player also can adapt the interaction-relationships by adding, deleting and updating interactions, obligations, conversations and operational parameters in the relationships. The organizer role presents management rights over its runtime model, for example, to a person who owns the model.

The organizer role exposes a management interface that contains methods for manipulating the structure and parameter of the runtime social interaction model such as addRole, deleteRole, addRelationship, deleteRelationship, addInteraction, deleteInteraction, addConversation, deleteConversation, addObligation, deleteObligation, addOperationalParameter, deleteOperationalParameter, bindRolePlayer, and unbindRolePlayer. By playing the organizer role, a human can perform adaptation manually using a graphical interface. On the other hand, automatic adaptation can be defined and performed through a computer programme or an agent or a set of predefined adaptation rules, as a player of the organizer role.
The following are two examples where changes of requirement are addressed using runtime adaptation:

**Example #1.** To add a new car, say Car#3, as a following car in the ConvoyDSIM, the following management methods (in the ConvoyDSIM Organizer) are required to be invoked (by the SocioTelematics client application) (see Fig. 6):

- `addSocialRole(“FollowingCar2”)`—to add another following car role to the ConvoyDSIM,
- `addSRelationship(“LeadingCar”, “FollowingCar2”, relName)`—to add a relationship between the LeadingCar and FollowingCar2 social roles,
- `addInteraction, addConversation, addObligation` and `addOperationalParam`—to add functional and non-functional information to the relationship as required, and
- finally, `bindRolePlayer`—to bind the third vehicle to the FollowingCar2 role.

**Example #2.** To assign a following car (say Car#3) to play the LeadingCar role due to the break-down of Car#1, the following management methods (in the ConvoyDSIM Organizer) are required to be invoked (by the SocioTelematics client application) (see Fig. 7):

- `unBindRolePlayer(“LeadingCar”, “Car#1”)`—to unbind the Car#1 from the LeadingCar role,
- `unBindRolePlayer(“FollowingCar2”, “Car#3”)`—to unbind the Car#3 from the FollowingCar2 role,
- `deleteRelationship(“LeadingCar”, “FollowingCar2”)`—to delete the interaction-relationship between the LeadingCar and FollowingCar2 roles,
- `deleteRole(“FollowingCar2”)`—to delete the FollowingCar2 role, and
- finally, `bindRolePlayer(“LeadingCar”, “Car#3”)`—to bind the Car#3 to the LeadingCar role.
7. Prototype implementation

We have implemented a prototype for SocioTelematics in the following three steps: (1) Model design, (2) Model deployment, and (3) Client application development, as illustrated in Fig. 8.

7.1. Tool support for modelling and deployment

The modelling tool for interaction-relationships is implemented using the Eclipse Modeling Framework (EMF)\(^1\) as a plug-in for Eclipse. The software developer can use this tool by dragging and dropping model components (i.e., roles, relationships) onto a canvas, specifying interactions between roles to build an interaction-relationships model (both player- and domain-centric) which is translated into an XML document.

In the model deployment step, this XML descriptor is instantiated using ROAD4WS\(^2\) which is a middleware extension to the Apache Axis2 web service engine. ROAD4WS exposes each role as a service and the associated interactions of the role as methods of that service. Once the model is deployed, the XML file is automatically picked up by ROAD4WS to instantiate both the functional and management part of the model. ROAD4WS creates the service interfaces (both the functional and management interface) for each and every role based on the functional and management part of the instantiated model. This service interfaces are exposed according to the WSDL 2.0 standard\(^3\). The service name is created by combining the model and social role name, following the pattern `modelname_socialrolename`. Each and every message signature of the social interactions is exposed as a WSDL operation. For example, all the interactions directed from the LeadingCar social role in the

---

\(^1\) [http://www.eclipse.org/modeling/emf/](http://www.eclipse.org/modeling/emf/)


\(^3\) [http://wsdl.apache.org/](http://wsdl.apache.org/)
Fig. 9. Leading Car social role (in ConvoyDSIM) related interactions are exposed as web service operations.

Fig. 10. Organizer (management) role (in ConvoyDSIM) operations are exposed as web service operations.

ConvoyDSIM, i.e., positionUpdate, routeUpdate and notifyMechIssue, are exposed as operations of the convoydsim_leadingcar service, as shown in Fig. 9. Similarly, all the state, behaviour and parameter related management operations are exposed as WSDL operations. Fig. 10 shows the partial management operations exposed as operation of the convoydsim_organizer service. These interfaces allow any WSDL 2.0 compatible client to invoke the operations.

ROAD4WS is able to (1) handle requests received from actors (here client applications); (2) check security settings for authorized access; (3) allocate requests into a message queue; (4) forward messages to roles; (5) evaluate conditions specified in the relationships and (6) send responses to relevant actors. Interactions between the model composites and those applications (as players) are supported by exchanging SOAP messages.

7.2. Client application development

We have implemented the SocioTelematics application for Android mobile devices. To support map, we use Google Map APIs for Android. To interact with the web server (platform) and process SOAP messages, we use ksoap2 [31]—a lightweight SOAP library for the Android platform. The application executes a number of tasks running as background services:

- Location sharing service—Listen to the location update from specified location provider (e.g., GPS, Network) and send the position to the other cars as positionUpdate interaction messages.
- Message fetching service—Pull the received messages from the queue in a proxy running on the server.
• **Message processing service**—Extract information from the fetched message, evaluate and fire coordination rules, and update the interaction history accordingly.

• **Map viewer service**—Periodically update the cars’ (both leading and following) positions on the map viewer so that drivers can see each other’s positions.

• **Text to speech conversion service**—To reduce driver distraction, this service provides voice notification (message name and contents) when a message comes.

The application has four fundamental panels such as **settings**, **convoy**, **interaction** and **adaptation** (see Fig. 11). When the application starts, the driver needs to configure the application using the **settings panel** (see Fig. 11(a)) by providing a user ID (to recognize the car) and password. Using the user’s credential, the application invokes the `getRoleList` organizer role function at the server side, fetches all social roles associated to its user and populates those roles in the main screen. Also based on each social role in the role list, the application invokes the `getInteractionList` function and accumulates associated interaction definitions for each social role. This means that the application is not tightly coupled with the social roles played by the car in a cooperative convoy and also the company from which the car has been rented. Thus, it serves the purpose of multi-modal use, i.e., without any internal logic change this application can be used in a car from any rental company as well as by a driver playing either the LeadingCar or FollowingCar social role.

Performing additional tasks in the cooperative convoy (e.g., forwarding information) may cause distraction. To facilitate collaboration but with less distraction, the application allows the driver to specify coordination rules for, say, forwarding information. In the **settings panel**, a driver can select an incoming interaction message and its corresponding outgoing message(s) as his/her coordination preferences. For example, the following car driver specifies a coordination preference as: “when a road block notification is received from the Roadside Assistance (as the `roadBlockNotification` interaction), forward it to the LeadingCar (as the `positionUpdate` interaction)”. Fig. 12 shows this coordination rule specified in the SocioTelematics application. Thus, when the application receives a message that matches with an incoming message in the coordination preferences, the application (as a coordinator player) sends the corresponding outgoing message(s) immediately on behalf of the driver. Also, the application provides voice notification (message name and contents) when a message comes.

Fig. 11(b) shows the **convoy panel** of the application. The map of the screen shows the cars’ positions in the cooperative convoy. The application uses GPS to show its car position on the map and notifies that position to others (as the `positionUpdate` interaction) using the location sharing service. The application gets the other car’s position once it receives a...
positionUpdate message from the other car. Also it shows a warning message as pop-up, for example, if the distance between the two cars violates their agreed distance.

The interaction panel (see Fig. 11(d)) of the application allows a driver to perform interactions and displays: the role list—associated with the driver of the car, the interactions list—list of interactions corresponding to the selected role in the role list, and the interaction log—interactions so far occurred.

The adaptation panel (see Fig. 11(d)) allows the leading car driver to perform adaptation on the ConvoyDSIM. The current version of the application provides an interface for the organizer player, e.g., the driver of the leading car, to perform/trigger three structural adaptations: create a new position (i.e., social role) for a new car to join in the convoy, join a car to the convoy and remove a car from the convoy.

As required, options to trigger other types of structural adaptation can easily be incorporated to the interface. The settings panel of the application interface allows the leading car driver to configure a number of parameters (i.e., performing parametric adaptations) such as position update frequency and maximum desired distance between two vehicles. Here, it is worth noting that using the application interface the organizer player only triggers or requests adaptation, while the actual adaptation is performed on the server side.

8. Experimental evaluation

In this section, we present the experimental results that demonstrate the feasibility and applicability of the SocioTelematics application with respect to the resource consumption and performance.

Section 8.1 presents our experimental testbed. Section 8.2 benchmarks the resource consumption of the SocioTelematics application on an Android Samsung Galaxy Tab phone. The results demonstrate that the application can be deployed on an off-the-shelf smart phone or Tablet with limited resource and computation power. Section 8.3 quantifies the performance of both the functional and management operations of the SocioTelematics application with respect to the communication latency between applications via the server. The results show that the application’s performance is acceptable in a cooperative convoy.

8.1. Testbed setup

We have set up an experimental testbed for evaluating the SocioTelematics system (illustrated in Fig. 13). Our implemented SocioTelematics system consists of a software suite running on Android smart phone/Tablet and cloud infrastructure. We use an Android Samsung Galaxy Tab phone (ARMv7 800 MHz processor, RAM 444 MB, External storage 4 GB) with 3G connection to run the client application.

Our Amazon cloud infrastructure is a standard windows based Apache Tomcat Web server that manages and mediates interactions by routing the messages, coming from the SocioTelematics application running in a car, to the destination endpoint. The middleware (ROAD4WS) running on the Tomcat Server takes the domain- and player-centric interaction-relationship models (XML descriptors) as input and dynamically generates web services with each social role being exposed as a service and each interaction term exposed as a method in the service. The middleware also generates a proxy for each of the players which stores all the incoming messages. Given the devices’ limited computation power and battery life—proxy service allows the player (device) to communicate asynchronously via SOAP interfaces.

We deployed the domain-centric and player-centric social interaction models related to the cooperative convoy scenario, i.e., BudgetDSIM, AVISDSIM, ConvoyDSIM, Car#1PSIM and Car#2PSIM, to the middleware platform. The middleware takes these social interaction models as input and dynamically generates Web services with each social role being exposed as a service and each interaction term exposed as an operation in the service. The middleware manages and mediates social interactions by routing the messages, coming from the SocioTelematics client application (i.e., player) running in a car, to the destination endpoint.

---

3 Because of lacking V2V/VANETs (Vehicular Ad Hoc Networks) infrastructure setup in real-world environment.

---

Please cite this article in press as: M.A. Kabir, et al., SocioTelematics: Harnessing social interaction-relationships in developing automotive applications, Pervasive and Mobile Computing (2014), http://dx.doi.org/10.1016/j.pmcj.2014.04.003
8.2. Evaluating resource consumption

We have profiled the resource consumption of the SocioTelematics prototype application on an Android Samsung Galaxy Tab, for CPU, memory (RAM), and storage. Fig. 14(a) and (b) show the application’s CPU and memory consumptions, respectively, over five minutes while the application was fetching messages from the middleware in every three seconds, updating Google maps view in every five seconds, and sending messages (i.e., performing interactions) once in every second.

Table 1 presents a comparison (average over a 5 min run) of SocioTelematics with two widely used similar (in terms of service used) Android Applications—Gmail and Google Maps. Gmail is an application that has a background service to interact with a web server for fetching and sending emails while Google Maps uses a location service to show position on the map view. This comparison is performed using the built-in task manager application of Samsung Galaxy Tab. The results in Table 1 show that SocioTelematics consumes reasonable resources and is competitive with existing applications in terms of resource efficiency.

8.3. Quantifying performance

We have examined both the functional and adaptation performance of the SocioTelematics using two cars in a cooperative convoy in 50 km of driving.

To quantify the functional performance, we have evaluated the communication latency between applications. Fig. 15(a) shows that the application takes an average 1100 ms to fetch a message from the server over 3G network. Fig. 15(b) shows that the server takes an average 250 ms to route a message from a proxy to another proxy via Car#1PSIM, ConvoyDSIM and Car#2PSIM, on the server. It is worth noting that the communication latency is highly variable and generally depends on the network traffic. In our testbed, we have also found that to pass a message from one client application to another client application takes around 1.6 s (given the fetching message interval is 1 s). This means that the application running in a car can display the other car’s position every 1.6 s which we believe is an acceptable time in a cooperative convoy scenario as it is not safety- or time-critical.

To quantify the application’s adaptation overhead, we have performed a number of adaptation operations, including add a car to the convoy and remove a car from the convoy. The results in Table 2 show that given the 1.12 s communication latency between the application and the server, on average the time to add and remove a car to and from the convoy at runtime takes 1.33 s (i.e., 1.121 + 0.209 s) and 1.167 s (i.e., 1.121 + 0.046 s), respectively. It takes around 0.42 ms to modify the maxDesiredDistance operational parameter. We believe these adaptation times are acceptable in a cooperative convoy as it is not safety- or time-critical.

9. Related work

There have been a number of attempts to design and develop automotive telematics systems. Industry originated initiatives such as Onstar [32] and BMW-Assist [33] have focused on safety rather than on user-centric convenience such as cooperative convoy. The most heavily advertised features of their telematics are lockout assistance, stolen vehicle location, and police notification upon air-bag deployment.

Please cite this article in press as: M.A. Kabir et al., SocioTelematics: Harnessing social interaction-relationships in developing automotive applications, Pervasive and Mobile Computing (2014), http://dx.doi.org/10.1016/j.pmcj.2014.04.003
Fig. 14. Resource consumption of the SocioTelematics while it was fetching messages from the middleware in every three seconds, updating Google maps view in every five seconds, and sending messages (i.e., performing interactions) once in every second.

Bisdikian et al. [34] have developed an open telematics platform using existing Web service interfaces and the Tiered Services over Public Wireless LANs (ts-PWLAN). Whilst the research shows some progress in bringing services into vehicular networks, the proposed platform involves tightly-coupled interaction between entities. Thus, it has limited support for adaptability and interoperability in such an open and dynamic environment.

Yu et al. [35] and Zhang et al. [36] focus on developing Open Service Gateway Initiative (OSGi) as a standard-based infrastructure for context-aware telematics. They adopt OSGi to connect internal and external vehicular networks (e.g., CANbus, MOST, Bluetooth, GPRS, etc.) and an ontology to model context information. The Sentient Car project [37] investigated an implementation of a vehicle-based sentient space where telematics systems are context-aware and adaptive. However, the approach relies on a tightly-coupled model of telematics systems, and thus has limited support adaptation. Wu et al. [38] developed ScudWare which is a semantic and adaptive middleware platform for smart vehicle space. ScudWare aims to support context-aware interaction and semantic integration between entities in a smart vehicle space by integrating a multi-agent technique with a context-aware and adaptive management service. Whilst ScudWare is rich in functionality, its architecture model is heavily loaded due to a multi-agent platform. Also, it is unclear how a semantic virtual agent is used to support a set of related tasks. Furthermore, ScudWare uses ontology to classify the properties of users, environment and vehicle, but does not take into account the interaction-relationships between these three entities, and their runtime changes.
(a) Time taken by the SocioTelematics client application (running on the Samsung Galaxy Tab) to fetch a message from the middleware (running on the Amazon cloud).

(b) Time taken by the middleware (running on the Amazon cloud) to route a message between Car#1 proxy and Car#2 proxy services through the appropriate running social interaction models.

Fig. 15. Communication latency.

Table 2
Time required to perform adaptation operations.

<table>
<thead>
<tr>
<th>Operations</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send an adaptation request from the SocioTelematics application to the Amazon server over a 3G network</td>
<td>1.121 s (±0.1 s)</td>
</tr>
<tr>
<td>Add a new car to the ConvoyDSIM at runtime</td>
<td>0.209 s (±0.013 s)</td>
</tr>
<tr>
<td>Remove a car from the ConvoyDSIM at runtime</td>
<td>0.046 s (±0.009 s)</td>
</tr>
<tr>
<td>Modify the maxDesiredDistance operational parameter in ConvoyDSIM</td>
<td>0.422 ms (±0.034 ms)</td>
</tr>
</tbody>
</table>

All of the above research assumes their users to be solitary and uniform, and makes no accommodation for their individual needs, or their social interaction-relationships. Compared to previous research, our interaction-relationship based approach emphasizes ‘socializing’ by supporting coordination and collaboration among entities (e.g., vehicles, service providers, infrastructures), externalizing interaction-relationships from the telematics application, modelling and implementing them separately using a service oriented paradigm, and managing their runtime adaptation.

10. Conclusions

We have presented SocioTelematics, a novel socially-inspired cooperative convoy telematics system that supports collaboration and coordination. Separating interaction-relationships from the application, and modelling and implementing
those using a service oriented paradigm, is the essence of our approach. This approach provides dynamic management of runtime adaptation. Furthermore, we provide a tool chain and client application for implementing such systems. Modification of the system can be done by developer’s changing the model using the provided tools and redeploying it on the server, and without any change in the application logic. Also, through an organizer interface, a user can modify the runtime instance of the model. In this way, our approach considerably reduces system upgrade and maintenance cost and allows the integration of the user’s requirements at runtime. We have tested the implemented prototype in real-life environments and evaluated the performance and resource consumption to demonstrate the system’s feasibility.

As future work, we aim to evaluate the usability of the application by collaborating with the Cooperative Research Centre for Advanced Automotive Technology (AutoCRC) and industry partners.

Acknowledgement

This work was supported in part by the Commonwealth of Australia, through the Cooperative Research Centre for Advanced Automotive Technology (AutoCRC).

References

vasive Computing and Communications Workshops (PERCOM Workshops), 2013, pp. 419–420. http://dx.doi.org/10.1109/PerComW.2013.852931.
2006.888386.
[8] M.A. Kabir, J. Han, A. Colman, J. Yu, Sociotelematics: leveraging interaction-relationships in developing telematics systems to support cooperative con-
MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments, pp. 164–182.
