SCaaS: A Platform for Managing Adaptation in Collaborative Pervasive Applications

Muhammad Ashad Kabir[1], Jun Han[1], Alan Colman[1], and Jian Yu[1,2]

[1] Faculty of Information and Communication Technologies,
Swinburne University of Technology, Melbourne, Australia
{akabir,jhan,acolman,jianyu}@swin.edu.au
[2] School of Computing and Mathematical Sciences,
Auckland University of Technology, New Zealand

Abstract. In this paper, we present a *social context as a service (SCaaS)* platform for managing adaptations in collaborative pervasive applications that support interactions among a dynamic group of actors such as users, stakeholders, infrastructure services, businesses and so on. Such interactions are based on predefined agreements and constraints that characterize the *relationships* between the actors and are modeled with the notion of *social context*. In complex and changing environments, such interaction relationships, and thus social contexts, are also subject to change. In existing approaches, the relationships among actors are not modeled explicitly, and instead are often hard-coded into the application. Furthermore, these approaches do not provide adequate adaptation support for such relationships as the changes occur in user requirements and environments. In our approach, inter-actor relationships in an application are modeled explicitly using social contexts, and their execution environment is generated and adaptations are managed by the SCaaS platform. The key features of our approach include externalization of the interaction relationships from the applications, representation and modeling of such relationships from a domain and actor perspectives, their implementation using a service oriented paradigm, and support for their runtime adaptation. We quantify the platform's adaptation overhead and demonstrate its feasibility and applicability by developing a telematics application that supports cooperative convoy.

1 Introduction

Mobile computing has brought a wireless revolution in recent years, enabling mobile internet access as an indispensable way of modern life. It has radically changed the way people perform tasks, access information and interact with one another. At the same time, the emergence of service-oriented technology and interoperability standards (e.g., Web Services) has made it possible to develop systems intended to support people's social activities and organizations' work. This trend has paved the way of a new breed of software systems that can mediate both tasks of individuals and collaborative tasks of a group in pervasive environments [1].

R. Meersman et al. (Eds.): OTM 2013, LNCS 8185, pp. 149–166, 2013.
© Springer-Verlag Berlin Heidelberg 2013

We view a collaborative pervasive environment (CPE) as an *interaction space* among users where users collaborate with each other towards a common objective by sharing information that has been provided by other participating actors. These actors could be users or services or sensors embedded in the environment. Such collaboration is subject to the *agreements* and *constraints* relevant to the *relationships* among actors, which are dynamic and need to *adapt* in response to the changes in user requirements and environmental factors.

Collaborative pervasive applications (CPAs), raise major challenges in terms of their development and management, including the dynamic aspect of them and their environments. The use of services [15] offers the possibility to design, deploy and manage these applications dynamically, providing the flexibility required.

Most of the approaches in the pervasive computing literature mainly focus on tasks of individual users, and provide very limited support to collaborative tasks of a group of actors [2]. Furthermore, existing approaches in developing collaborative pervasive applications (e.g., [4–8]), and middleware architectures and frameworks for pervasive computing [14] are limited in supporting the dynamic *relationships* between actors, which themselves are an important aspect of context. In particular, there is a lack of support for managing the *adaptation* in such relationships in response to *changes* in the requirements and environments.

This research explores the concept of *social context* as a means to represent the relationships among actors and presents a *platform* to provide support for *adaptation* by managing such social contexts and their changes, in a service-oriented manner. Social context in computing is often used to refer to the people, groups and organizations that an individual interacts with [3]. Taking this view, we define *social context* as a representation of the interactions among the relevant actors. That is, social context defines the *constructed relationships* between social roles, and these relationships *define* and *constrain* the interactions between the actors playing those roles. To model social context, we employ Role-Oriented Adaptive Design (ROAD) [16] among many different approaches to design role-based software systems using *agent* paradigm, as ROAD brings a number of design principles that support flexible management and runtime adaptations. One of the key principles of ROAD is separation of *functional* and *management* operations. We model social context from two perspectives: domain-centric and player-centric [12]. A *domain-centric social context (DCSC)* model captures a collaborative view of the interaction relationships among the actors whereas a *player-centric social context (PCSC)* model captures an actor's coordinated view of all its interactions (across domains).

In this paper, we present a SCaaS platform for developing collaborative pervasive applications from their *high-level* specifications (represented in terms of DCSCs and PCSCs), and for managing their *adaptations* to cope with *runtime* changes. Figure 1 presents an overview of the SCaaS platform. SCaaS takes the DCSC and PCSC models (specified by an application designer) as inputs, and *instantiates* these models by generating *management* and *interaction* interfaces (as Web Services) for applications to invoke. Thus, using applications (running on mobile devices) actors can interact with each other and manage their social

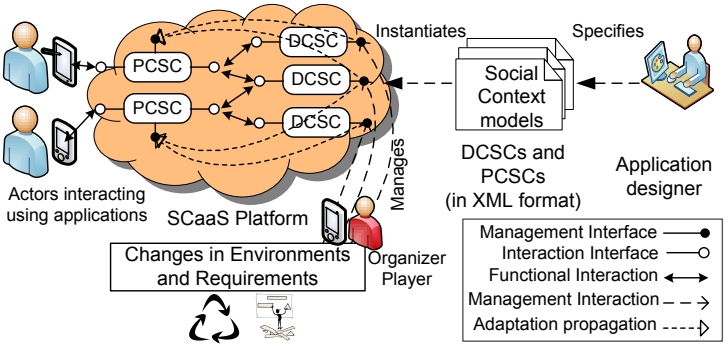


Fig. 1. Overview of the *SCaaS* platform

contexts. In particular, we focus on how the SCaaS platform supports runtime adaptation in social contexts to cope with changes in the evolving user requirements and environmental factors. In relation to the SCaaS platform, this paper makes the following four major contributions: (1) we identify different types of changes and the various adaptations to cope with such changes, and we propose management operations and present possible ways to perform adaptations using these operations, (2) we introduce *states* of the social context and its elements to enable adaptation in a safe manner, (3) we propose a *protocol* for adaptation propagation from DCSCs to PCSCs, and (4) Finally, we implement a *SCaaS platform* for creating the *execution* environment of social contexts and managing their runtime *adaptations*, and quantify the platform's *adaptation overhead*.

The paper is organized as follows. Section 2 presents a scenario where applications need to interact, collaborate and adapt in pervasive environments. After giving an overview of our social context models in Sect. 3, we present the SCaaS platform for managing adaptations in social contexts in Sect. 4. Section 5 discusses its prototype implementation, while Sect. 6 presents the experimental evaluation and a case study. After reviewing related research in Sect. 7, we conclude the paper in Sect. 8.

2 Application Scenario

Consider that two groups of tourists in two cars hired from two different rental companies want to drive together from Melbourne to Sydney. The car rental companies provide different types of support to their customers based on the customers' insurance policies, service availability, and so on. Let us assume that the two cars, Car#1 and Car#2, are rented from Budget and AVIS respectively.

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 information (acquired from the service providers and infrastructure systems) with other vehicles. Such interactions are subject to defined agreements and constraints among the entities (i.e.,

vehicle to vehicle, vehicle to service providers, and vehicle to infrastructure). For instance, 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. These two cars have access to different types of services: Car#1 has access to a Travel Guide Service (TGS) while Car#2 has access to a real time Traffic Management Service (TMS). 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 1000m. 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 gets the updated route plan from its TGS by specifying its preferences (e.g., avoid the route with blocked road) and notifies that route information to Car#2. Both cars notify each other of their positions every 10 seconds. If either vehicle experiences mechanical problems (e.g., flat tyre, engine issue) it needs to notify the other vehicle.

Applications facilitating the cooperative convoy need to fulfill two major requirements. **First**, the applications should support interactions complying with the *agreed* interaction relationships (i.e., constraints and obligations). For the drivers to perform the additional tasks (e.g., forwarding information) may cause distraction and have undesirable consequences. Thus, to facilitate collaboration with less distraction, the applications need to provide a coordinated view of the interactions, allow drivers to specify their coordination preferences and perform the coordination in an automated manner. **Second**, the applications need to support runtime adaptation as the interaction relationships evolve over time, and need to *adapt* with the *changes* in requirements and environments. For instance, a mechanical problem of the leading car may require it to handover the leading car role to one of the following cars (assuming there are multiple following cars). Because of heavy rain, the maximum distance may need to be reduced from 1000m to 600m. 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.

To address the first requirement, in our previous work [12], we have proposed an approach to modeling interaction relationships from both the domain and player perspectives. The DCSC model allows the interactions associated with a domain such as Budget or AVIS or Cooperative Convoy to be captured, while the PCSC model provides an overall view of all the interactions of a particular individual (e.g., driver of Car#1) and allows coordination among its interactions.

To address the second requirement, in this paper, we propose the SCaaS platform where the DCSC and PCSC are the basis of this platform. At runtime, the interaction relationships captured by DCSCs need to *adapt* with the *changes* in user requirements and environments. However, the PCSCs are dependent on DCSCs. The SCaaS platform *manages* both the DCSCs and PCSCs, and their dependencies in a consistent manner by supporting the adaptations in the DCSCs and the *adaptation propagation* from DCSCs to PCSCs as changes occur.

3 Social Context Models: An Overview

In this section, we briefly discuss the DCSC and PCSC, and illustrate how these social context modeling perspectives allow us to capture the above cooperative convoy scenario. A detailed discussion can be found in [12].

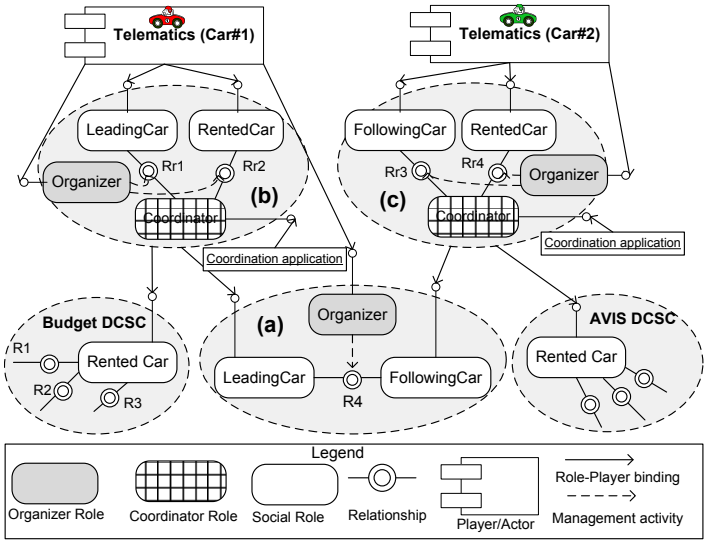


Fig. 2. Social context models for a cooperative convoy: (a) ConvoyDCSC, (b) Car#1PCSC, (c) Car#2PCSC

3.1 Domain-Centric Social Context Models

The *Domain-Centric Social Context* (DCSC) model captures the relationships among social roles associated with a particular domain or environment such as a company, a cooperative convoy, and so on. A DCSC model comprises of four key elements: social role, relationship, player, and organizer role. A *social role* represents the expected functional interactions of a participating actor with respect to the social context. Social roles are loosely-coupled elements and are modeled as first class entities, and as such they are separated from their players (e.g., actors) who play those roles. A *relationship* is an association between two social roles, which represents the interactions and interdependencies between those roles (or their corresponding players or actors). It mediates the interactions between social roles by defining what functional *interactions* can occur between the social roles and the sequences of these interactions (named *conversations*). In addition, a relationship also defines the *non-functional* requirements of the interactions in terms of *operational parameters* and *obligations* (e.g., time constraints on interactions) imposed on the players associated with that relationship. The *organizer role* and its player provide the capability for *managing* and *adapting* a social

context to cope with *changes* in the requirements and dynamic environments. While social roles, players and relationships are entities at the social context's functional layer, the organizer role and its player are entities at the social context's management layer.

In the above scenario, there are three domains that need to be modeled, namely, the two car rental companies (Budget and AVIS) and the cooperative convoy. According to the agreements between the drivers of the two cars, the *ConvoyDCSC* model (see Fig. 2a) consists of two roles: *LeadingCar* (LC) and *FollowingCar* (FC), and their interactions are captured in the R4 (LC-FC) relationship (see Table 1) where an interaction is represented using a message signature and a direction of the message (i.e., AtoB, BtoA or both). Figure 3 shows an XML representation of the $i8$ interaction. Also, there may be other constraints and behavioral properties (e.g., conversations, obligations (e.g., o1 in Table 1)), but for simplicity we do not include all of them. As Car#1 plays the leading car role, it is also designated to play the organizer role of the *ConvoyDCSC* model. So, by playing the organizer role (through an application interface), the leading car driver can change the model (e.g., add, delete or update roles and relationships) at runtime. In a similar way, we can also model *BudgetDCSC* and *AVISDCSC* where Car#1 and Car#2 play the *RentedCar* role. Due to page limit, we do not present the details of these models which can be found in [17].

Table 1. Partial description of R4 (LC-FC) relationship in *ConvoyDCSC*

Specification from scenario	Notational representation
FC sends ahead road blocks information to the LC	i7:{notifyRoadBlock, FCtoLC, ack}
LC updates the route information to the FC	i8:{routeUpdate, LCtoFC, ack}
Both cars notify each other of their positions every 10 seconds	i9:{positionUpdate}//bi-directional o1:{i9,Time,periodic,=,10,seconds}
One car notifies mechanical problems to the other car	i10:{notifyMechanicalIssue}
Either vehicle may leave the convoy	i11:{leaveConvoy}
Maximum distance between the LC and FC is 1000m	p1:{maxDistance=1000m}

```
<tns1:Interaction name="routeUpdate" id="i8">
        <Direction>LCtoFC</Direction>
        <Parameters>
                <Parameter>
                        <Type>String</Type>
                        <Name>routedetails</Name>
                </Parameter>
        </Parameters>
        <Return>String</Return>
</tns1:Interaction>
```

Fig. 3. XML representation of the $i8$ social interaction

3.2 Player-Centric Social Context Models

In addition to the common view of a domain-centric social context model, an actor may have its own perception or view of the domain with respect to the role(s) it plays and the interactions it participates in that domain. Moreover, an actor may operate in different domains. Thus, we also model the social context from an actor's perspective, namely *Player-Centric Social Context* (PCSC). The PCSC model provides an overall view of all the interactions of an individual (across different domains) and allows coordination of its interactions.

Fig. 2b and Fig. 2c show the player-centric models of Car#1 and Car#2 respectively (and how they relate to the domain models). Like a DCSC, a PCSC contains social roles, actors/players and an organizer role. In addition, it contains a *coordinator role* and *role-centric relationships*. In the PCSC, all social roles are played by the actor (application) in question and are connected with the coordinator role through the role-centric relationships. The *coordinator role* is a means of achieving inter-domain coordination, i.e., interactions in one domain can be used for interactions in another domain. An actor can coordinate its interactions explicitly through the application's user interface or it may define some rules or use an intelligent application to coordinate its interactions on its behalf [12]. A *role-centric relationship* is the aggregation of all the relationships associated with a particular *social role* in a DCSC model, but localized in the player-centric model. For example, the $Rr2$ role-centric relationship in the PCSC of Car#1 is the aggregation of $R1$, $R2$ and $R3$ in the *BudgeDCSC*.

4 The SCaaS Platform

4.1 Social Context at Runtime

Social contexts are not just a modeling or design-time construct, and they are also *runtime* entities that *mediate* runtime interactions between actors. Interactions among an actor, its player-centric social context model, and the relevant domain-centric social context models are *loosely* coupled and use a messaging style. A *(runtime) social context* acts as a message *router* that (1) receives messages from an actor, (2) evaluates conditions specified in associated relationships, and (3) passes the messages to another actor or a social context (as a player) or notifies the actor(s) in case of any condition violation. For the PCSC, all the incoming and outgoing messages are intercepted by the *coordinator role* which is played by the actor's *coordination application*. The application coordinates messages on behalf of the actor based on her preferences.

SCaaS facilitates the runtime realization of social contexts for supporting mediated interactions between collaborative actors, such as those of Car#1 and Car#2. However, the *requirements* of such applications are subject to continuous *change*. Thus, social contexts need to be *managed* and *adapted* to ensure the proper functioning and evolution of the applications in which the social contexts play a part. In the rest of this section, we discuss the management and adaptation support provided by the SCaaS platform in offering social context as a service.

4.2 Types of Changes

In general, there are two types of changes that require adaptation in a social context as well as across social contexts.

Changes in Environments. During convoy, it may start to rain heavily or the cars may move from one jurisdiction to another which operates different traffic management systems. Such changes cause social context adaptation and are referred to as *changes in environments*.

Changes in Requirements. A third vehicle could join when the convoy is already on the way; a broken-down following car might leave the convoy before reaching its destination; or the leading car might have a mechanical problem which requires to handover the leading car role to one of the following cars (assuming multiple following cars). Such situations also cause adaptation and are referred to as *changes in requirements*.

4.3 Adaptations in a Social Context

Runtime adaptation, often called *dynamic* adaptation, is a widely used term and is extensively studied in multiple disciplines. In pervasive computing, this term is used to denote any kind of modification at the running phase of the system [19]. In general, such runtime adaptation can be classified into two categories: *parameter* adaptation and *structure (compositional)* adaptation [18]. The adaptation in a social context to cope with the changes in user requirements and environments also can be of these two types: structural (compositional) and parametric.

We achieve *structural* adaptation in two ways:

- *Modifying topology* – Adding and removing social roles, players and relationships are carried out. For instance, a third vehicle could join the convoy when it is already on the way, or a broken-down following car leaves the convoy before reaching the destination. These situations lead to the addition or removal of roles, players and relationships in the *ConvoyDCSC*.
- *Modifying the binding between a social role and its player* – The same social role can be played by different players at different times. The *binding* between the role and the players is dynamic. For instance, in the *AVISDCSC*, the traffic management role can be played by different traffic management systems in the convoy at different times as the vehicle moves from one jurisdiction to another. Also due to a mechanical problem of the leading car (Car#1), the *Car#1PCSC* needs to unbind from the leading car role in the *ConvoyDCSC* and one of the following cars can be assigned to play the leading car role by binding that car's PCSC to the leading car role in the *ConvoyDCSC*.

We achieve *parametric* adaptation through the *modifying relationships* where adding, removing or updating *interactions*, *obligations*, *conversations* and *operational parameters* are carried out. For instance, in the *ConvoyDCSC*, the *maxDistance* parameter value in *R4* relationship may be required to be reduced, from 1000m to 600m because of heavy rain.

Management Operations and Adaptation Rules. The organizer role provides the management capability of a social context. The principle of separation between a role and its player is also applied to the organizer role. The organizer role is internal to a social context and allows its player to manage both the *structure* and *parameters* of the social context. The organizer role presents management rights over a runtime social context model, for example, to an actor who owns the runtime model. The organizer role exposes a management interface that contains methods for *manipulating* the structure and parameter of the runtime social context model such as *addRole, deleteRole, addRelationship (addRel), 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 program* or an *agent* or a set of predefined adaptation *rules* as a player of organizer. For example, the *maxDistance* parameter value can be reset to 600m using the following Event-Condition-Action (ECA) rule:

```
adaptation-rule "Update maximum distance"
when
 EnvironmentChangeEvent(name=="RainingStatusValueChanged") //Event
then
 if(rainingStatus == HEAVY_RAIN) //Condition
 callMethodInOrgInterface("updateOperationalParam("R4",maxDistance,600m)")
```

Social Context States and Safe Change. To perform adaptation in a safe manner without affecting the message flow and loss of messages, we maintain the state of each entity, i.e., social role, relationship and runtime social context as a whole. Figure 4 shows the states and their transitions. When a social context is deployed all of its entities enter into *Idle* state. When a conversation is started, the associated social roles and relationship move to the *Active* state and remain there until the conversation completes. A social context enters into *Active* state when any of its social roles or relationships becomes *Active* and remains there until all of its roles and relationships become *Idle*. When an adaptation operation (structural or parametric) starts, the entity enters into the *Reconfiguration* state.

The time when a change cannot be made is the moment when the entity is in *Active* state. For instance, an adaptation operation cannot be performed in a social role or relationship when a conversation (request/response) associated with these entities is in progress. In that case, the adaptation request will be buffered and executed in the future after the entities enter into the *Idle* state.

4.4 Adaptations across Social Contexts

As stated in the previous section, a PCSC provides a coordinated view of all the interactions of an individual across different domains, and an individual plays

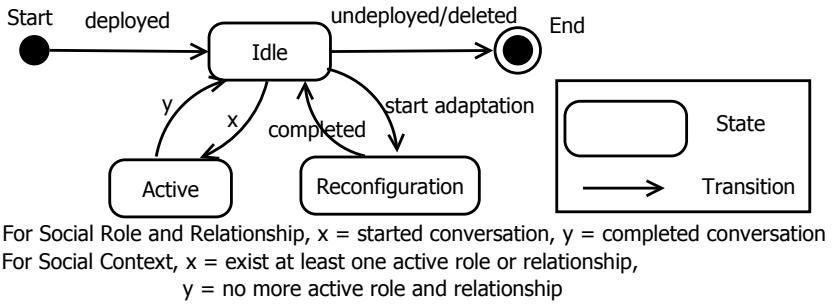


Fig. 4. State machine for *Social Role*, *Social Relationship* and *Social Context*

roles in multiple domains/DCSCs through her PCSC. Thus, an adaptation in a DCSC should be propagated to its corresponding PCSCs. Figure 5 shows a basic protocol for such propagation. In this protocol, the DCSC organizer triggers the adaptation in a PCSC by invoking the following methods: *triggerRoleAcquisition*, *triggerRoleRelinquishment* and *triggerUpdateRelationship*.

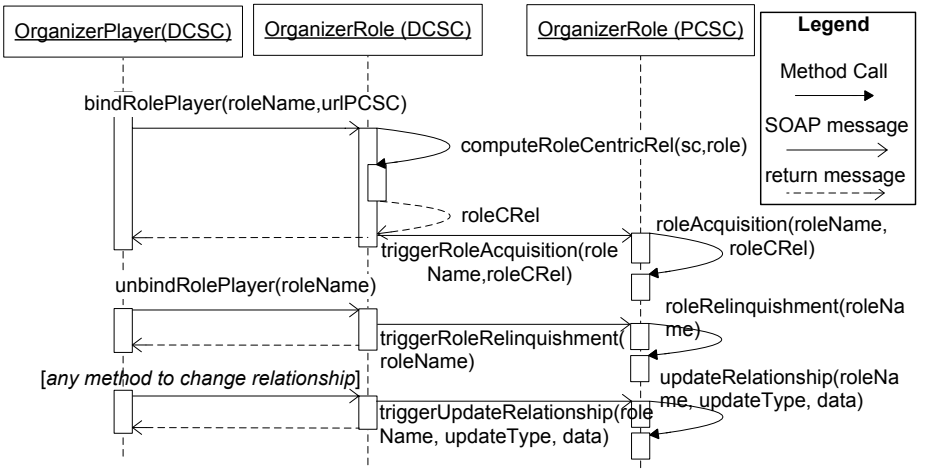


Fig. 5. Cross-DCSC/PCSC adaptation propagation

The adaptations across social contexts have three aspects:

1. *Binding a player to a social role in the DCSC* – When a player binds to a role in a DCSC, her PCSC should add that role and its role-centric relationship. Thus, for the *bindRolePlayer* request, the DCSC invokes *computeRoleCentricRel* method to compute the role-centric relationship of a particular social role which is the aggregation of all the relationships associated with that

Algorithm 1. Computing Role-centric Relationship

```
1: procedure COMPUTEROLECENTRICREL(sc, r)        ▷ r is a social role and sc is a social context
2:     roleCentricRel ← empty                    ▷ create an empty relationship
3:     relList ← getAllRel4Role(sc, r)           ▷ relationships connected to r
4:     for all rel ∈ relList do                  ▷ rel is a relationship in the relList
5:         for all i ∈ rel do                    ▷ i is an interaction in rel
6:             roleCentricRel.addInteraction(i)
7:         end for
8:         for all c ∈ rel do                    ▷ c is a conversation in rel
9:             roleCentricRel.addConversation(c)
10:        end for
11:        for all o ∈ rel do                    ▷ o is an obligation in rel
12:            roleCentricRel.addObligation(o)
13:        end for
14:        for all p ∈ rel do                    ▷ p is an operational parameter in rel
15:            roleCentricRel.addOperationalParam(p)
16:        end for
17:    end for
18:    return roleCentricRel
19: end procedure
```

social role in the DCSC (see Algorithm 1). Then the DCSC invokes the *triggerRoleAcquisition* method in the PCSC organizer with the social role and its role-centric relationship, as parameters. The PCSC organizer executes *roleAcquisition* method to adapt its structure by adding a social role and relationship based on the received information.

2. *Unbinding a player from a role in the DCSC* – When a player is unbound from the DCSC, her PCSC should be adapted by removing that role and its role-centric relationship. Thus, for the *unbindRolePlayer* request the DCSC invokes the *triggerRoleRelinquishment* method with the social role name as the parameter. The PCSC organizer executes *roleRelinquishment* method to adapt its structure by deleting the social role, and the relationship between that social role and the coordinator role, when those entities are in *Idle* state (i.e., safe to delete).

3. *Updating a relationship in the DCSC* – All the updates in a relationship and/or a social role in a DCSC should be propagated to the corresponding PCSC(s). Thus, for any modification request in a relationship (i.e., to add, delete or update an interaction, conversation or obligation), the DCSC organizer invokes the *triggerUpdateRelationship* method in the PCSCs which are bound to the associated social roles in that relationship. Then the PCSC organizer executes the *updateRelationship* method to reflect the changes.

4.5 Revisiting the Scenario

Let us consider a situation that requires management and adaptation in a social context and across social contexts which can be addressed using the SCaaS platform.

If the leading car (Car#1) breaks-down, the *ConvoyDCSC* organizer player (the leading car driver) invokes (through a user interface) the *unbindRolePlayer* method to unbind *Car#1PCSC* from the LC and then the *ConvoyDCSC* (organizer) invokes the *triggerRoleRelinquishment("LC")* method in *Car#1PCSC*. As a result, the *Car#1PCSC* updates its structure by deleting the LC role and the relationship between the LC and Coordinator role (see Fig. 6ⓐ). Furthermore, to assign a following car (say Car#2) to play the leading car role, the *ConvoyDCSC* organizer player invokes the *unbindRolePlayer("FC", urlCar#2PCSC)* followed by *bindRolePlayer("LC",urlCar#2PCSC)* to first unbind *Car#2PCSC* from the following car role and then bind it to the leading car role. As a result, the *ConvoyDCSC* organizer invokes the *triggerRoleRelinquishment* and *triggerRoleAcquisition* methods respectively which ultimately updates the *Car#2PCSC* by deleting the FC role and its associated relationship (see Fig. 6ⓐ) followed by adding the LC role and its associated relationship (see Fig. 6ⓑ).

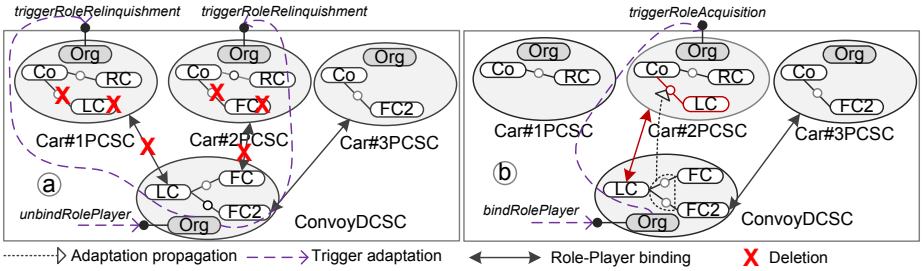


Fig. 6. Runtime adaptation in social context models due to break-down of leading car

5 Prototype Implementation

We have implemented the SCaaS platform by adopting and extending the ROAD4WS [13] which is an extension to the Apache Axis2[1] web service engine for deploying adaptive service compositions. The SCaaS platform exploits JAXB 2.0[2] for creating DCSCs and PCSCs runtime from their XML descriptors. JAXB helps the generation of classes and interfaces of runtime models automatically using an XML schema. The platform exposes each *social role* as a *service*, the associated *interactions* of the role as operations of that service.

[1] http://axis.apache.org/
[2] http://jcp.org/en/jsr/detail?id=22

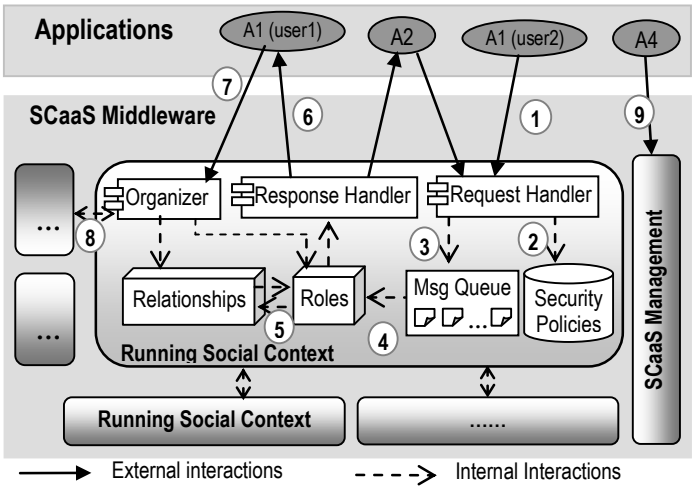


Fig. 7. The SCaaS platform architecture

The *conversation* and *obligations* specified in the *social relationship* are evaluated as *event-condition-action* rules and implemented using Drools[3]. Actors (players) playing the roles invoke the operations which create messages. Such messages are routed to other players who they are collaborating with. As illustrated in Fig. 7, the SCaaS platform generates *runtime social contexts* which are able to (1) handle requests received from players (i.e., applications); (2) check security settings for authorized access; (3) allocate requests into a message queue; (4) forward messages to corresponding social roles; (5) evaluate conditions specified in the relationships; (6) send requests to relevant players. A runtime social context also can (7) receive a management request (from a user/application) and adapt itself accordingly, and (8) propagate the adaptation to other social contexts as necessary. The SCaaS Management module handles (9) platform level management requests such as create, delete, deploy and undeploy social contexts as required by the user/application. Interactions between the runtime models and their players (i.e., external interactions) are supported by exchanging SOAP[4] messages.

The *runtime adaptations* are supported by the Java reflection mechanism and the Drools engine. To cope with the changes in environments and requirements, at runtime, Javassist[5] allows generation of *new* classes and *modification* of existing classes, which helps to add new social roles/relationships and change existing roles/relationships, respectively. Drools engine allows the SCaaS to inject new rules and delete existing rules from the working memory which facilitates the addition and deletion of conversations, obligations and parameters in the

[3] http://www.jboss.org/drools/
[4] http://www.w3.org/TR/soap/
[5] http://www.jboss.org/javassist

relationships. This way of implementation provides flexible and easy runtime adaptation in a particular entity of a social context without interrupting the other entities of that social context.

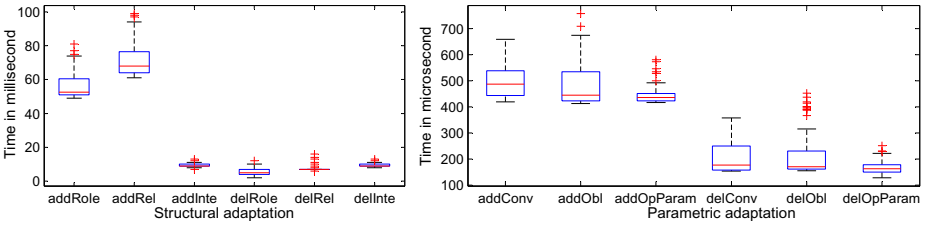
6 Experimental Evaluation and Case Study

The goal of our experiment had to quantify the SCaaS platform's *adaptation overhead*. We installed the SCaaS platform on a machine with Core i3 2.2 GHz CPU, 8GB RAM and Windows 7 OS. We used Java 1.6, Drools 2, Tomcat 7.0.21 and Axis2 1.6.1 in this experiment. As a case study, we also implemented the motivating scenario as a proof-of-concept application, and measured the application's adaptation overhead in real-life experiment to demonstrate the feasibility and applicability of the SCaaS-based application.

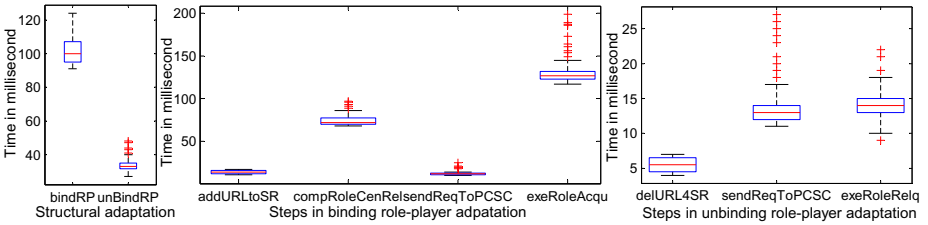
6.1 Adaptation Overhead

To evaluate the adaptation overhead, we deployed 100 DCSCs where each DCSC is consisted of 10 roles connected in a ring topology using 10 relationships, and each relationship is comprised 6 interactions, 6 conversations and 6 obligations. Once we deployed 100 DCSCs to the SCaaS platform, SCaaS created 10 PCSCs for 10 players where each player plays a role in each of the 100 DCSCs. Thus, each PCSC contained 100 social roles and 100 role-centric relationships. We executed each of the structural and parametric adaptation operations (e.g, addRole, addConv) 1000 times over the 100 DCSCs. We measured the time from the moment the adaptation was requested, to the moment Axis2 updated the services. The box plots in Fig. 8 show the summary of the results where the horizontal line inside each of the boxes represents the median (average time). The results show that the deletion operations (e.g., *delRole*, *delRel*, and *unbindRP*) take less time compared to the addition operations (i.e., *addRole*, *addRel*, and *bindRP*). Figure 8a shows the time required to perform different structural and parametric adaptations in a social context. The results show that the structural adaptations take more time than the parametric adaptations. Among the structural adaptation operations, adding a relationship (*addRel*) in a social context takes the longest time, around 68 millisecond (ms) (on average), as it needs to update the configuration of two social roles, where as deleting a social role (*delRole*) takes the least time, around 5ms. For different parametric adaptation operations, the required time is related to rule injections and deletions in the Drools engine and lies between 162 and 486 microseconds.

The box plots in Fig. 8b illustrate the adaptation overhead results across a DCSC and a PCSC. The leftmost figure shows the total time required for the *bindRP* (bind role-player) and *unbindRP* (unbind role-player) structural adaptations. The middle figure shows the time required for each step in the *bindRP* adaptation, including the time to add a URL to a social role (*addURLtoSR*), to compute a role centric relationship (*compRoleCenRel*) using Algorithm 1, to send a request to a PCSC (*sendReqToPCSC*), and to execute the role acquisition method (*exeRoleAcq*). The rightmost figure shows the time required for each



(a) Adaptation in a social context



(b) Adaptation across social contexts

Fig. 8. Adaptation overhead

step in the *unbindRP* adaptation, including the time to delete an URL from a social role (*delURL4SR*), to send a request to a PCSC (*sendReqToPCSC*), and to execute the role relinquishment method (*exeRoleRelq*). The results show that on average the *bindRP* and *unbindRP* operations take 100ms and 33ms, respectively, which we believe is an acceptable overhead in collaborative applications.

6.2 Case Study

To demonstrate the real-world applicability and feasibility of our approach, we have developed an adaptive collaborative application on top of the SCaaS platform, called *SocioTelematics* that enables multiple cars to form cooperative convoys. This application allows the drivers to see each other's positions on the Google Maps. Using this application, drivers in the convoy can adapt and manage their social contexts and interactions. The SCaaS makes it easy to develop this application based on their supposed interactions and without worrying about the underlying message communication and the evaluation of the messages, as the runtime support and adaptation of these social contexts and interactions are externalized to and managed by the SCaaS platform. Moreover, the runtime adaptation capability provided by the SCaaS platform allows the application to respond to changes in requirements and environmental factors, without any change in the application code.

We evaluated the application's adaptation overhead using two cars in a cooperative convoy over 50 kilometres of driving where the SCaaS platform was deployed in the Amazon EC2 and the two client *SocioTelematics* applications were running on two in-car Android Samsung Galaxy Tabs with 3G connections. The results in Table 2 show that given the 1.12 second communication latency

Table 2. Time required to perform adaptation operations

Operations	Time
Send an adaptation request from the *SocioTelematics* application to the Amazon server over a 3G network	1.121 sec
Add a new car to the *ConvoyDCSC* at runtime, i.e., *addRole*, *addRel* and *bindRP*	0.209 sec
Remove a car from the *ConvoyDCSC* at runtime, i.e., *unbindRP*, *delRel* and *delRole*	0.046 sec
Change the $o1$ operational parameter in $R4$ relationship	0.422ms

between the application and the server, on average the time to add and remove a car to and from the convoy at runtime take 1.33 sec (i.e., 1.121+0.209) and 1.167 sec, respectively, which we believe are acceptable times in a cooperative convoy.

7 Related Work

7.1 Platform for Collaborative Pervasive Applications

The need for supporting collaboration in pervasive computing environments has emerged in recent years (e.g., [4], [8]). Such research has focused on collaborative interactions between different types of actors such as user-user and device-device, for various purposes. The SAPERE [4] middleware exploits social network graph to establish collaboration for sharing data among spatially collocated users devices. CoCA [5] is a ontology-based context-aware service platform for sharing of computational resources among devices in a neighbourhood. Both SAPERE and CoCA focus on collaboration among *devices*, where SCaaS focuses on collaboration among *users*. Similar to SCaaS, MoCA [6], a middleware architecture for developing context-aware collaborative applications, focuses on collaboration among *users*. But unlike SCaaS, the collaborations among users in MoCA are not based on *predefined* goals or tasks, rather driven by spontaneous and occasional initiatives. CASMAS [7] and UseNet [8] focus on collaborative activities among users to achieve a common goal like SCaaS. But none of them explicitly *model* the interactions among users.

Moreover, all of the above approaches lack support for *managing* the dynamicity and complexity of the social context as highlighted in this paper. The relationships between actors and their adaptations are not modeled explicitly, and instead are often *hard-coded* directly into the applications. To the best of our knowledge, there is no work to date that addresses the runtime adaptation of social context models in response to the changes in requirements and environments.

7.2 Middleware Support for Runtime Adaptation

Much research has been carried out into middleware support for runtime adaptation in context-aware systems (e.g., MADAM [9] and 3PC [10]) and

service-oriented systems (e.g., MUSIC [11] and MOSES [20]). These middleware solutions mainly target the tasks of individual users/applications and have focused on reconfiguring applications' settings (rather than interaction relationships) based on physical context information (e.g., place, time)/quality of service requirements (e.g., performance, reliability), rather than interaction relationships. Moreover, their proposed runtime models are application-specific and cannot be used to model interaction-relationships among collaborative actors.

In contrast to these solutions, the SCaaS platform targets collaborative pervasive applications, and focuses on executing adaptation by explicitly realizing interaction relationships using social contexts and providing an organizer interface to change such social contexts. On the other hand, SCaaS does not address the monitoring of environment changes (i.e., physical context information), analyzing such information or making adaptation decisions. In that sense, the SCaaS middleware is not a substitute for existing middleware solutions that manages physical context information, rather can be built on top of those solutions as appropriate, in order to manage (as a service) social interactions and context adaptation for collaborative pervasive applications.

8 Conclusion

We have presented a novel *Social Context as a Service* platform for supporting application-level adaptations and enabling mediated-interactions among actors (individuals with their applications) in collaborative pervasive environments. Our approach *externalizes* interaction-relationships from the application implementation, explicitly *models* the interactions in terms of social contexts, *separates* functional interactions from management operations, and provides runtime realization of social contexts. All these facilitate the systematic *management* of dynamic interaction-relationships between actors and support their *adaptation* to cope with the *changes* in user requirements and environments.

SCaaS facilitates both *structural* and *parametric* adaptations in social contexts which are realized through the *management* (organizer) interface of the social contexts. SCaaS also maintains the inherent *dependencies* among social contexts and keeps them consistent through coordinated *cross-social context adaptation*. Our model-driven approach and service-oriented implementation make it easier to develop different adaptive collaborative applications on top of SCaaS. We have quantified the adaptation overhead of the SCaaS platform through an experimental evaluation and demonstrated its applicability with a cooperative convoy telematics application.

References

1. Conti, M., et al.: Looking ahead in pervasive computing: Challenges and opportunities in the era of cyber-physical convergence. Pervasive Mob. Comput. 8(1), 2–21 (2012)

2. Sancho, G., et al.: What about collaboration in ubiquitous environments? In: Proc. of 10th Int. Conf. on New Techn. of Distr. Syst., NOTERE (2010)
3. Endler, M., et al.: Defining Situated Social Context for pervasive social computing. In: Proc. of PerCom Workshop (2011)
4. Castelli, G., Rosi, A., Zambonelli, F.: Design and implementation of a socially-enhanced pervasive middleware. In: Proc. of PerCom Workshop, pp. 137–142 (2012)
5. Ejigu, D., et al.: CoCA: A Collaborative Context-Aware Service Platform for Pervasive Computing. In: Proc. of ITNG, pp. 297–302 (2007)
6. Sacramento, V., et al.: MoCA: A Middleware for Developing Collaborative Applications for Mobile Users. IEEE Distributed Systems (2004)
7. Cabitza, F., et al.: CASMAS: Supporting Collaboration in Pervasive Environments. In: Proc. of PerCom, pp. 286–295 (2006)
8. Rodriguez, I.B., et al.: A model-driven adaptive approach for collaborative ubiquitous systems. In: Proc. of the AUPC Workshop (2009)
9. Geihs, K., et al.: A comprehensive solution for application-level adaptation. Softw. Pract. Exper. 39(4) (2009)
10. Handte, M., et al.: 3PC: System support for adaptive peer-to-peer pervasive computing. ACM Trans. Auton. Adapt. Syst. 7(1) (2012)
11. Rouvoy, R., et al.: MUSIC: Middleware Support for Self-Adaptation in Ubiquitous and Service-Oriented Environments. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems. LNCS, vol. 5525, pp. 164–182. Springer, Heidelberg (2009)
12. Kabir, M.A., Han, J., Colman, A.: Modeling and Coordinating Social Interactions in Pervasive Environments. In: Proc. of 16th Int. Conf. on Eng. of Complex Compu. Syst. (ICECCS), pp. 243–252 (2011)
13. Kapurge, M., Colman, A., King, J.: ROAD4WS - Extending Apache Axis2 for Adaptive Service Composition. In: Proc. of the EDOC (2011)
14. Raychoudhury, V., et al.: Middleware for pervasive computing: A survey. Pervasive and Mob. Comput. (2012)
15. Papazoglou, M.P., van den Heuvel, W.-J.: Service-Oriented Architectures: Approaches, Technologies and Research Issues. VLDB J. 16(3) (2007)
16. Colman, A., Han, J.: Roles, Players and Adaptive Organisations. Applied Ontology: An Interdisciplinary Journal of Ontological Analysis and Conceptual Modeling 2, 105–126
17. Kabir, M.A., et al.: SocioTelematics: Leveraging Interaction-Relationships in Developing Telematics Systems to Support Cooperative Convoys. In: Proc. of Ubiquitous Intelligence & Comput., pp. 40–47 (2012)
18. McKinley, P.K., Sadjadi, S.M., Kasten, E.P., Cheng, B.H.C.: Composing adaptive software. Computer 37, 56–64 (2004)
19. Kakousis, K., Paspallis, N., Papadopoulos, G.A.: A survey of software adaptation in mobile and ubiquitous computing. Enterp. Inf. Syst. 4 (2010)
20. Cardellini, V., et al.: MOSES: A Framework for QoS Driven Runtime Adaptation of Service-Oriented Systems. IEEE Transactions on Software Engineering 38, 1138–1159 (2012)