Context-Based Access Control for Ridesharing Service

Nikolay Teslya, Alexey Kashevnik, Michael Pashkin
SPIIRAS
St.Petersburg, Russia
{teslya, alexey, michael}@iias.spb.su

***Abstract*—The paper describes a context-based access control model for a ridesharing service. Ridesharing is a shared use of a car by the driver and one or more passengers for a joint trip. The service is based on the smart space concept. For this purpose the Smart-M3 platform is used. Currently the Smart-M3 platform doesn't have an appropriate access control mechanism meeting the following requirements: supporting a flexible, descriptive and well-defined policy language and taking into consideration the context information. Therefore, the usage of the context-based access control model has been proposed. This model is built as a combination of the role-based and attribute-based access control models. It uses roles, which are assigned dynamically based on the user's context, and meets the requirements to the access control in the smart space. An analysis of information transfer through the ridesharing service modules is used for defining the user's context. The model has been implemented within an access control broker, which controls the access to the smart space resources.**

***Keywords*—Ridesharing, Smart Space, Context, Access Control.**

I. Introduction

The ridesharing service provides a real-time fellow-travelers search. It makes mobile devices of drivers and potential passengers as well as computational devices interoperate in a common smart space, based on the Smart-M3 platform [1]. The computational devices have a special software (e.g., ridesharing brokers) for complex computations, which cannot be run on the users' devices for several reasons, such as energy saving and computation capabilities. To operate the service needs information from users, such as their paths and preferences for searching matching paths. Most of this information cannot compromise the user's privacy, but there can be some information that has to be protected with access control for being used only by defined persons and services. For example, people don't often want to share their locations to others. At the current state of the Smart-M3 platform development, all of the information in the smart space is available for reading by any of its participant without any restrictions.

The key idea of the Smart-M3 platform is device, domain, and vendor independence. Devices and software entities can publish their embedded information for other devices and software entities through simple, shared information brokers. Information exchange in smart space is implemented via HTTP, using Uniform Resource Identifier (URI). The Smart-M3 platform consists of two main parts: information agents and a kernel. The kernel consists of two elements: Semantic Information Broker (SIB) and data storage. The information agents are software entities, installed on mobile devices of the smart space users. These agents interact with SIB through the Smart Space Access Protocol (SSAP) using "insert", "update", "remove", "query", and "subscribe" transactions. The SIB is the access point for receiving the information to be stored, or retrieving the stored information. All this information is stored in the data storage as a graph, that conforms the rules of the Resource Description Framework (RDF). In accordance with these rules all information is described by triples "Subject – Predicate – Object".

Nowadays there are three ongoing projects addressing information security for the Smart-M3 based smart space. The first project is based integrated mechanisms of triples protection in the Smart-M3. This protection only works for transactions, which modify the triples [2]. It means that participants can protect triples to be inserted, updated or removed, according to a special template. However all of the triples in smart space still can be queried or subscribed.

The next project grants access permissions to triples like a file system grants permission to folders and files. It is achieved via mapping of all triples to a virtual file system and using the filesystem's possibilities of access control [3], [4]. Triples are mapped to files and the triples hierarchy mapped to the folder structure. Users are identified and authorized in the smart space through host identity protocol (HIP) [5], [6]. This approach provides rather flexible access control. At the same time, it is quite difficult to define user groups and to configure access permissions for all the files and folders since the project uses the discretionary access control model, which becomes very complicated in a dynamic system like a smart space with a lot of users and resources.

The last project aims to provide a mechanism to grant access permissions based on the user's context [7]. The following scheme of secure access to smart space resources has been proposed. The participant is identified by the system when registered in the smart space. The unique identifier is generated and saved in the access control broker. The public and private keys are generated using the RSA algorithm. A consumer service (the service of the participant) sends request to the public smart space to access some private information from service provider and subscribes to the corresponding response about the access granting. The smart space service provider accepts the request and calls a special access control broker service for the access permission. The access control

broker confirms that this consumer is authenticated and applies the rules from the security policies to assign the role to the participant. The access permission is granted based on the role of the participant and then is sent to the smart space service, which requested it, through the virtual private space. Usage of the context for assigning roles makes the model more flexible and appropriate for services in the smart space.

The rest of the paper is organized as follows. Section 2 describes the ridesharing service. Section 3 presents basic elements of the context-based access control model such as a context, participant roles, main model rules, and the model scheme. Section 4 provides a configuration of the access control model for the ridesharing service.

II. RIDESHARING SERVICE

As it has been mentioned earlier, the ridesharing service helps to search fellow-travelers for a joint trip [8]. It is based on the idea of joint traveling and provides automation of the joint trip search. The ridesharing service receives various data from smart space and finds overlaps in users paths, interests, and preferences. Using this information the service constructs a set of instructions for each user, which consists of possible fellow-travelers, meeting points, meeting time, travel cost and other.

The ridesharing service consists of two main parts: a client application and ridesharing broker. The interaction between these parts is achieved through the smart space, based on the Smart-M3 platform.

The client application can be installed on user's mobile devices with Android operating system, which is one of the most popular mobile operating systems in world at the moment. The main functions of the client application are:

- Collecting various information from the user and sending it to the smart space.
- Receiving information about joint trip from the smart space and presenting it to the user.

The client application's working scenario is shown in Fig. 1. There are two main sources of route information. First is

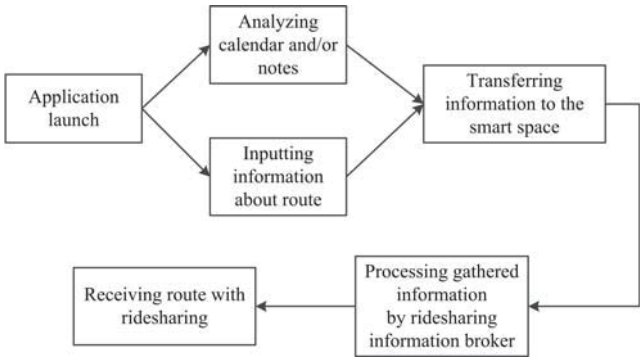

Fig. 1. Client application working scenario

the user's calendar. After the application has been started once, it analyzes all existing information about meetings, places to visit, etc, depersonalizes it and publishes it to the smart space. If there is no information in the calendar then application waits for a user input. The user should input the information about his/her planned route, which is then processed and transferred to the smart space, where it is used for finding a fellow-traveler. When the fellow-traveler is found, the user receives (through the subscribe transaction) information about possible joint trip and can agree with the trip or decline it.

The ridesharing broker is a software, installed on computers with high computation capabilities. Its main function is receiving information from the smart space and finding matches between ridesharing service's users paths. Finding such matches is a very complicated task that needs a lot of computational power and not energy safe. Therefore, this task is performed on a specialized computers.

For the service interoperability an ontology, describing main entities has been introduced. The macro level ontology is based on integration of components of the mobile devices' ontologies (Fig.2).

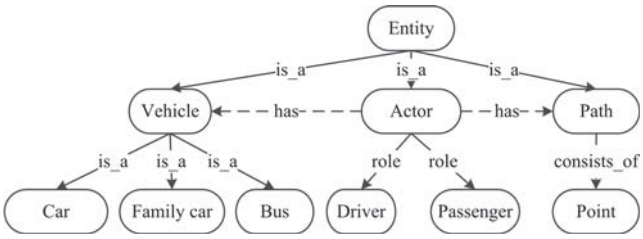

Fig. 2. Ridesharing service ontology on the macro level

The ridesharing service ontology consists of three main components: vehicles, actors and paths.

A. Vehicles

The vehicles are:

- cars with no more than four vacant seats;
- family cars with 5 to 8 vacant seats;
- buses with 9 and more vacant seats.

B. Actors

The actors are: drivers and passengers. All of them have vehicles and paths. For example, driver has his/her own car and several points defining his/her home, work and other locations. Passenger may prefer some vehicle type and has points of home, work, and other locations. The entity class *"actor"* consists of (Fig. 3):

- *ID*. Unique ID for each user;
- *Name*. First and last name of the user;
- *Point*. Path point belong to the user (at least two: the start and the end);
- *Delay*. Maximal possible time of waiting in the meeting point.

The entity class *"Driver"* is a subclass of the class *"Actor"* and inherits all its properties with two own properties:

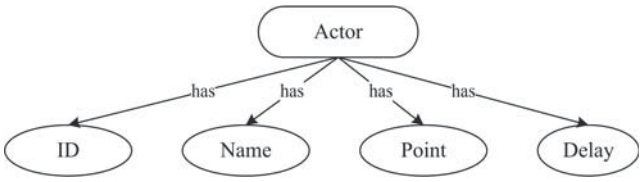

Fig. 3. Entity class “Actor”

- *Vehicle*. Vehicle type;
- *Detour*. Maximal detour from the shortest path.

The class *“Passenger”* is a subclass of the class *“Actor”* and inherits all its properties with own property *“Detour”* the same as in the class *“Driver”*.

For the path definition the set of points is used. This set is an ordered list of key points obtained as a result of the shortest path searching algorithm (e.g., Dijkstra or A*). The class *“Point”* has the following structure (Fig. 4):

- *PreviousPoint*. Contains the previous path point. For the start point its value is “FALSE”;
- *Latitude*;
- *Longitude*;
- *DriveByVehicle*. If the point belongs to the passenger, it contains the driver who gives a ride to this passenger. If the passenger walks then its value is “FALSE”;
- *VacantSeats*. The number of vacant seats in vehicle in point;
- *VacantItemPlace*. The number of vacant places for cargo items;
- *Date*. Date, when the user will be at this point;
- *Time*. Time, when user will be at this point;
- *Wait_time*. How long the user will be waiting in this point.

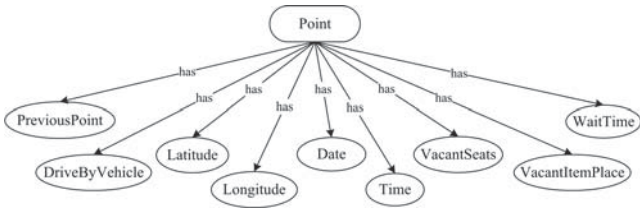

Fig. 4. Entity class “Point”

Since the ontology in the Smart-M3 is represented in RDF standard, user description in smart space looks like follows:

```
('user1', 'name', 'Arthur P. Dent')
('user1', 'is_a','Driver')
('user1', 'vehicle','Car')
('user1', 'social_network 'Facebook')
('user1', 'social_network_id,'a.dent')
```

```
('user1', 'point','user1point1')
('user1point1', 'x', '60.0363')
('user1point1', 'y', '30.3677')
('user1point1', 'date', '01.10.2013')
('user1point1', 'time', '10.00')
('user1point1', 'vacantseats', 2)
('user1point1', 'vacantitemplaces', 3)
```

```
('user1', 'point','user1point2')
('user1point2, 'x', '59.80726')
('user1point2', 'y', '30.38291')
('user1point2', 'date', '01.10.2013')
('user1point2', 'time', '12.42')
('user1point2', 'vacantseats', 2)
('user1point2', 'vacantitemplaces', 3)
```

This example shows a situation in the ridesharing service, when the user ‘Arthur P. Dent’ with a page in the Facebook social network, drives by car from *point1* to *point2* at the defined time and ready to pick up two passengers, which moving in the same direction. In [9] the logistics service ontology is described in detail.

III. Context-based access control model for the smart space resources

According to [10] the security and privacy in the smart space should support a flexible, descriptive and well-defined policy language, and be able to take into consideration a context information. The context is defined as any information that can be used to characterize the situation of an entity, where an entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves [11].

The context-based access control model, proposed in [7], meets these requirements. It is built based on the combination of the role-based (RBAC) and attribute-based access control (ABAC) models. These models use roles and attributes as a base for the access granting decision. The roles are static and set up by a system administrator. The context-based access control model is closer to the ABAC model, but also uses roles, which are assigned dynamically based on the user trust level and help to manage access to the resources. The trust level calculation is based on the participant’s context, which includes attributes, identifying the user (user ID and public key); user location; current date; device, which requests the informatio;, etc. A special smart space service implemented within the access control broker has been proposed for this model. This service grants access to the resources for the smart space services guided by the security policies. According to this model the public information can be published to smart space and processed by all participants, but the private information is provided only for appropriate participants through virtual private spaces when the corresponding access permissions are granted [7].

The context of the smart space participant consists of the physical and virtual components (Fig. 5). The physical component is due to the fact that each participant in the smart space is also represented in the physical environment, which requires the processing of its properties from that environment.

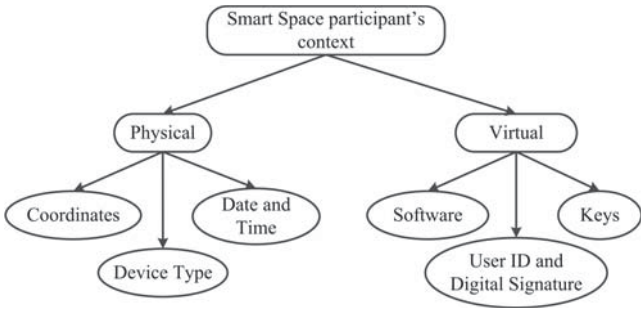

Fig. 5. Components of smart space participant's context

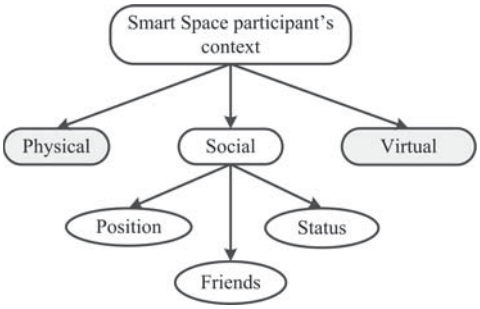

Fig. 6. Components of smart space participant's context in case of participant is a member of the Web-community

The physical component includes: geographical location of the device, date and time, type of a device. Using this information, the smart space services can determine the current user location, and time of the information access. It enables granting different access permissions from different user locations.

Virtual component of the context is due to the fact that each smart space participant interacts with others, and is characterized by a set of attributes that characterize it in a smart space. This component includes software used by the participant for accessing the smart space, digital signature (the participant's identifier and the identifier encoded by the private key), and public key. This information enables authentication and authorization of the participant and provides encoding of the private data. For the Web-community the participants add a social component to the context (Fig. 6). This component includes, for example, position in the company, social relationships. The social component of the context enables granting access to the employees at different positions with the different trust levels, some private data can be shared only between friends, etc. All components of the context are collected and stored on the smart space devices. They become available upon the request of the access control broker.

General scheme of request process is presented below in Fig. 7. In this scheme the smart space consists of: a participant, which requests the information; a service, which provides this information; and an access control broker, which provides access permissions to the participant based on its context. The information flow between the participant and the service becomes private due to the virtual private space, which has no intersections with the public space. The participant publishes the request to the public smart space for accessing private information (in the RDF notation) and subscribes to the corresponding response about the access granting. The smart space service accepts this request and calls the access control broker for the access permission decision making.

The access control broker reads the participant's context and verifies its digital signature using the open key. If the signature is correct, the broker confirms that this user is authenticated and applies the rules from the security policies to assign the role to the participant. The access permission is granted based on the role of the participant and then is sent to the smart space service, which requested it. If the access to the resource is granted, the smart space service creates a virtual private space. The information requested by the participant is transferred to this private smart space. The connection information (space IP, space port and space name) is encrypted via the open participant's key and is sent to the public smart space. If the access is denied, the service sends the corresponding notification to the smart space participant.

Participant, who sends the information request, gets the notification via the Smart-M3 subscription mechanism. If the access is granted the participant decodes the encoded data with its private key and creates a connection to the specified virtual private smart space. When the requested information is transferred the virtual private space is destroyed.

IV. Access control model for ridesharing service

The example of the user description in the ridesharing service from section II includes private information about that user. First, there are no people who want to be tracked. Without the access control, every user can collect locations of other users and predict the future steps for different purposes. Also, there are no people who want to share their real names and social networks information for everyone. Collecting any information about any person is also restricted by law of any state. The access control broker for ridesharing service has been developed to prevent the collecting information about users of this service by criminals.

The following conceptual model of the ridesharing service, which includes the access control broker, has been proposed (Fig. 8).

The ridesharing service consists of: participants, which are drivers and passengers; ridesharing brokers, which search for matching paths between participants; and access control broker, which grants or denies access to participant's information based on the context of the requesting user.

A. Defining context

The user's context can be defined using the ontology of the ridesharing service parts. This subsection describes the main context components and provides information about how these components can be used in the context-based access control model.

1) Physical part: The physical part of the user's context consists of:

- *User's Location* consists of latitude and longitude and allows building routes for ridesharing and defining

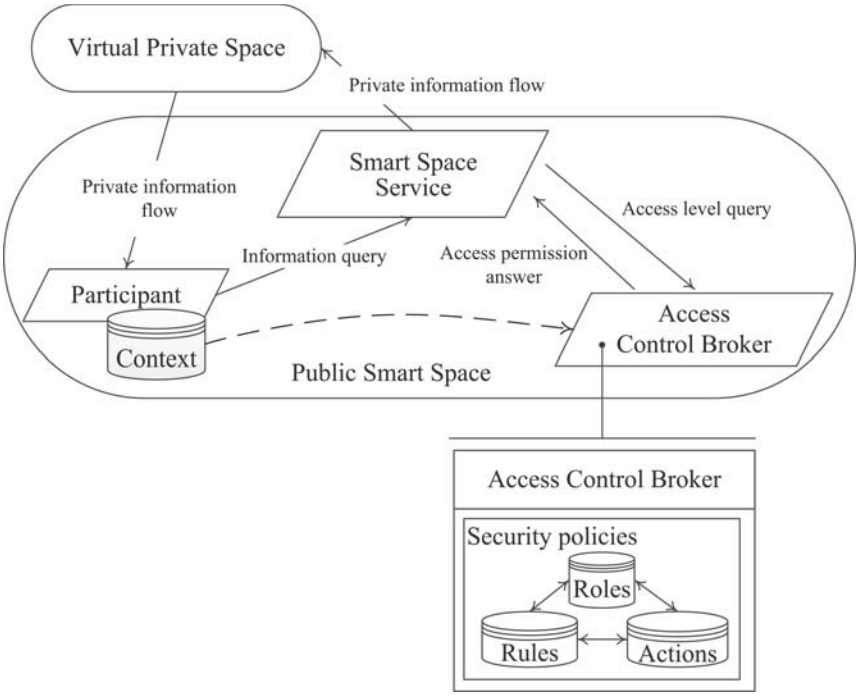

Fig. 7. General scheme of context-based access to Smart space resources

place, which the user tries to access the information from;

- *Date and Time* allow to provide access permission and search fellow-travelers in specified time interval .

2) Virtual part: The virtual part of the user's context consists of:

- *ID* used as a part of identifier of the user in the service.
- *User's authentication information*, such as password, private key, certificate, etc. This is a main component in any authentication protocol. Software and device fingerprint can also be used as a part of authentication information, because of several combinations of the software configuration are unique and all mobile devices have unique IMEI code and most of them have unique hardware configuration.

3) Social part: The social part of the user's context mostly collected from different social networks, such as Facebook, VKontakte, Google+ and consists of:

- *Role in service* that can be driver or passenger. Allows to provide information only for specified roles and search fellow-travelers.
- *Information from the social networks* that includes the common information about user, such as name, surname, date of birth, birth place, common interests, etc. On its base the service can use filters defined by users, for example, filter users who are not younger than 20 years old or have more than 50% merging in interests.

B. Rules definition

The participant's context is used to define the trust levels assigned with its role. Usage of the user roles allows to simplify policies and make them human-readable and easy to configure. Each component of the context is associated with the trust level. This level is represented by a number in the range $[0, 1]$ and depends on the context of the current situation. For example, the trust level of '0.1' and '0.9' can be assigned for access from the different user's locations respectively. It means, that the user, who lives, for example, in the Saint Petersburg, Russia can grant access to the private information for users in the nearest regions (Russia and Finland) who will have high trust level (0.9) and restrict it for users from other, farthest regions (China, North Korea) who will have low trust level (0.1).

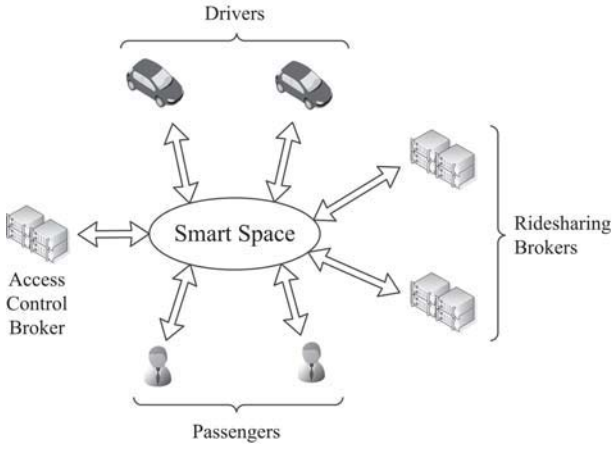

Fig. 8. Conceptual model of ridesharing service with access control

The logical function taking into account trust levels of all appropriate context components is used to assign a role to the participant. Proposed ridesharing service has five main roles: owner, trustedPassenger, untrustedPassenger, trustedDriver, untrustedDriver. The trustedPassenger and trustedDriver are separate roles, because in this case they can control information access for other passengers and drivers. For example, the driver can show his/her information only to passengers, not to the other drivers. If the user has untrusted role, then he/she can access only the common information, which cannot compromise any other user. The owner is a person who sharing the information, so he can read, write and edit it.

1) TrustValue rules: These rules are used to assign the numeric trust value to the context component. The following example shows rules of this type:

```
TrustValue(isPassenger == true) = 1;

TrustValue(isPassenger == false) = 0;

TrustValue('08:00' < currentTime <
'17:00') = 0.8;

TrustValue('08:00' > currentTime >
'17:00') = 0.2;

TrustValue(currentLocation 'in set'
[Russia, Finland]) = 0.9;

TrustValue(currentLocation 'in set'
[China, North Korea]) = 0.1;

TrustValue(commonInterests 'more than'
1/2) = 0.7;

TrustValue(commonInterests 'less than'
1/4) = 0.3;

TrustValue(birthDateAccept 'before' 1985)
= 0.8;

TrustValue(birthDate 'after' 1985) = 0.2;

etc.
```

Most of the rules are set as user's preferences in his/her profile while configuring the application. In this way the user shares information about his/her preferences with other service participants using the mobile application. The access control broker interprets this information and makes a decision about granting access permission for other users based on the predefined rules. Also, this information is used by ridesharing brokers to find the best path matches, according to the user's interests and preferences.

The trust level values are set by the security service and based on the estimations of the security service provider's experts according to the features of the particular smart space service.

2) AssignRole rules: These rules are used at the time of access request. They have a form of logic equations. The following example presents the situation when the role *"trustedPassenger"* is assigned only if the user is a passenger, his/her current location is Russia or Finland, the time is from 8 am to 5 pm and his/her birth date is before 1985. If the user doesn't satisfy this rules then the next rule is checked.

```
AssignRole(trustedPassenger) =
(TrustValue(isPassenger) = 1 &
(TrustValue(currentLocation ∈ (0.7, 1))
& (TrustValue(currentTime) ∈ (0.6, 1))&
(TrustValue(birthDateAccept) ∈ (0.7, 1)).

AssignRole(untrustedPassenger) =
(TrustValue(isPassenger) = 0 &
(TrustValue(currentLocation ∈ (0.1, 0.5))
& (TrustValue(currentTime) ∈ (0, 0.6))&
(TrustValue(birthDateAccept) ∈ (0, 0.7)).

etc.
```

The same rules are defined for the driver role.

3) Permission rules: These rules contain access control policies, which determine whether a participant with a certain role is allowed to access a particular resource type or not. These rules are also set up by the users in their preferences.

```
Permission(trustedPassenger) =
"readCommon", "readPrivate";

Permission(untrustedPassenger) =
"readCommon".

etc.
```

C. Access control broker implementation

The access control broker for ridesharing service has been developed using Python programming language. As an example, situation of two users, trying to get access to the information about each other, is described. There are following rules for access permission granting: users should be friends or have a mutual friends and both should be in the same region.

First, it is needed to configure the *TrustValue* rules in the access control broker. In Python it looks as follows:

```
trustValue_rules =
{'friendship': {'friend':0.9,
'not_friend':0.1},
   'is_a': {'Driver':0, 'Passenger':1},
   'currentLocation':{Russia:0.8,
```

```
Finland:0.8, China:0.1, North Korea:0.1}
}
```

Then, the ranges for assigning roles and allowed actions for role actions need to be configured:

```
ranges_for_roles =
{'trustedUser':
   {'friendship':[0.8,1], 'is_a':[1],
   'currentLocation':[0.7,1]},
'untrustedUser':
   {'friendship':[0,0.79], 'is_a':[0],
   'currentLocation':[0,0.7]},
}

role_actions =
{'trustedUser':
   ['read_private_inf','read_only_public'],
   'untrustedUser': ['read_only_public'] }
```

The main functionality of the access control broker has been realized via the following algorithms. The first one is a *"formalize_context"* algorithm. It is used for collecting user's context and processing it to the trust levels (Algorithm 1). This algorithm works by the following way: it extracts context components from the smart space and other possible sources (like a device information, social networks, etc.) and applies rules assigning trust levels to the each context component. The result of algorithm is a map (an array of *"key:value"*) with corresponding context component as a *key* and it's trust value as a *value*.

Algorithm 1 formalize_context

Require: {*user1_information* $\neq \emptyset$
and *user2_information* $\neq \emptyset$
and *trustValue_rules* $\neq \emptyset$}
context = {}
for all *key* in *trustValue_rules* **do**
  if *key* == "friendship" **then** {Special feature}
    if *user1_information*["social_network_id"] in
    user1_information[*friend_list*] **then**
      context["friendship"] =
      trustValue_rules["friendship"]["friend"]
    else
      context["friendship"] =
      trustValue_rules["friendship"]["not_friend"]
    end if
  else
    context[*key*] =
    trustValue_rules[*key*][*user1_information*[*key*]]
  end if
end for
return *context*

The second algorithm is a *"set_role"* (Algorithm 2). It assigns roles to the user based on the output of the *"formalize_context"* algorithm and the *"AssignRole"* rules, defined before. The result of this algorithm is a set of user roles which are satisfy rules.

Algorithm 2 set_role

Require: {*user_context* $\neq \emptyset$
and *ranges_for_roles* $\neq \emptyset$}
user_roles = []
for all *key* in *ranges_for_roles* **do**
  OverallFlag = **true**
  for all *key2* in *ranges_for_roles*[*key*] **do**
    OverallFlag = *OverallFlag* **and**
    (*ranges_for_roles*[*key*][*key2*][0] $<$ *user_context*[*key2*])
    and
    (*ranges_for_roles*[*key*][*key2*][1] $>$ *user_context*[*key2*])
    if *OverallFlag* **then**
      user_roles.append(*key*)
    end if
  end for
end for
return *user_roles*

The third algorithm checks the permissions, which belong to the role, and compares them with the permissions needed to access the queried information (Algorithm 3). If this permission matches, then the permission is granted and user who requests the access can read this information.

Algorithm 3 set_actions

Require: {*user_role* $\neq \emptyset$
and *role_actions* $\neq \emptyset$}
allowed_actions = []
for *role* in *user_role* **do**
  allowed_actions =
  list(set(*allowed_actions*) $\cup$ set(*role_actions*[*role*]))
end for
return *allowed_actions*

V. Conclusion

In the first versions of the smart space ridesharing service, the information sharing has been implemented without any restrictions. For the wide usage of the service, it was needed to provide a mechanism that allows ridesharing service users to restrict access for their private information. This paper describes such mechanism that uses the context-based access control model. This model allows users to share their private information only with persons they trust via using the easy-configurable preferences. The ontology of the ridesharing service has been used for defining the context of the service users. Rules for each group of users has been configured and roles have been defined according to the model. These roles allow providing flexible access control to the user's private information.

Acknowledgment

The presented results are part of the research carried out within the project funded by grants # 13-07-12106, 13-01-00286, 13-07-00336, 13-07-00039, 13-07-13159, 13-07-12095, and 13-07-00271 of the Russian Foundation for Basic

Research and grant # 12-04-12062 of the Russian Foundation for Humanities.

REFERENCES

[1] J. Honkola, H. Laine, R. Brown, O. Tyrkko, "Smart-M3 Information Sharing Platform", *in Proc. IEEE Symp. Computers and Communications (ISCC'10)*, IEEE Comp. Soc., 2010, pp. 1041-1046.

[2] A. D'Elia, J. Honkola, D. Manzaroli, and T. Cinotti, "Access control at triple level: specification and enforcement of a simple rdf model to support concurrent applications in smart environments", *in Smart Spaces and Next Generation Wired/Wireless Networking*, 2011, pp. 63-74.

[3] K. Yudenok, "Smart-M3 Security Model,". *in Proc. of 11th Conf. of Open Innovations Assoc. FRUCT*, Apr. 2012, pp. 210–211.

[4] K. Yudenok, I. Nikolaevskiy "Smart-M3 Security: Authentication and Authorization Mechanisms,". *in Proc. of 13th Conf. of Open Innovations Assoc. FRUCT*, Apr. 2013, pp. 153–162.

[5] A. Gurtov, M. Komu, and R. Moskowitz, "Host Identity Protocol (HIP): Identifier/locator split for host mobility and multihoming", *in Internet Protocol Journal*, vol. 12, no. 1, Mar. 2009, pp. 27-32.

[6] R. Moskowitz, P. Nikander, P. Jokela, T. Henderson, "RFC 5201: Host Identity Protocol", Web: http://tools.ietf.org/html/rfc5201

[7] A. Smirnov, A. Kashevnik, N. Shilov, N. Teslya, "Context-based Access Control Model for Smart Space", *in 5th Int. Conf. on Cyber Conflict (CyCon 2013)*, June 2013, pp. 47-62.

[8] A. Kashevnik, N. Teslya, N. Shilov, "Smart Space Logistic Service for Real-Time Ridesharing", *in Proc. of the 11th Conf. of Open Innovations Association FRUCT*, Apr. 2012, pp. 53-62.

[9] A. Smirnov, A. Kashevnik, N. Shilov, H. Paloheimo, H. Waris, S. Balandin, "Smart Space-Driven Sustainable Logistics: Ontology and Major Components", Sergey Balandin, Andrei Ovchinnikov (eds.), *in Proc. of the 8th Conf. of Open Innovations Framework Program FRUCT*, Nov. 2010, pp. 184-194.

[10] J. Al-Muhtadi, A. Ranganathan, R. Campbell, M.D. Mickunas, "Cerberus: a context-aware security scheme for smart spaces", *in Pervasive Computing and Communications. (PerCom 2003). Proc. of the 1st IEEE Int. Conf.*, 2003, pp. 489-496.

[11] A.K. Dey, D. Salber, and G.D. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications," *Context-Aware Computing, A Special Triple Issue of Human-Computer Interaction*, vol. 16, 2001.