

# Smart Places: Multi-Agent based Smart Mobile Virtual Community Management System

Muhammad Fahad, Olivier Boissier, Pierre Maret, Néjib Moalla, Christophe Gravier

▶ **To cite this version:**

Muhammad Fahad, Olivier Boissier, Pierre Maret, Néjib Moalla, Christophe Gravier. Smart Places: Multi-Agent based Smart Mobile Virtual Community Management System. Applied Intelligence, 2014, 41 (4), pp.1024-1042. 10.1007/s10489-014-0569-2 . hal-01015456

**HAL Id: hal-01015456**
**https://hal.science/hal-01015456**

Submitted on 15 May 2017

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Smart Places: Multi-Agent based Smart Mobile Virtual Community Management System

Muhammad Fahad[a], Olivier Boissier[b] Pierre Maret[c], Nejib Moalla[a] and Christophe Gravier[c]
[a]*DISP lab, Universite de Lyon, IUT Lumiere lyon 2, 160 Boulevard de l'universite, 69676 Bron, France*
[b]*Ecole Nationale Supérieure des Mines, FAYOL-EMSE, LSTI, F-42023 Saint-Etienne, France*
[c]*Université de Saint-Etienne, 10 rue Tréfilerie, F-42023 Saint Etienne, France*

**Abstract.** Now-a-days the advancement in mobile device technology aims to build complex computational systems providing maximum level of flexibility, decentralization, simplest form of interactivity, and ease of use. Recently, the launch of agent-oriented platform *JaCaMo* and its Android client based platform **JaCa-Android** provide an appropriate level of abstraction to build smart mobile client server systems providing these attributes. By using these platforms, we have developed a multi-agent based **Smart Mobile Virtual Community Management System** (*SMVCMS*) that makes possible to install a decentralized and open management of virtual communities. This paper addresses the design and architecture of our multi-agent server and client application. It elaborates different features of our system, such as how a participant of virtual communities is supported by a *Jason* agent that encapsulates the logic and the control of the participation to a virtual community (such as publishing posts, notifying members, recommendation for the user, etc.). It discusses how the set of **CArtAgO** artifacts provides the basic functionalities and operations giving access to the functionalities for knowledge exchange in virtual communities, and personal agents on *Android* exploits those artifacts to execute their tasks while achieving their individual and collective goals. We have employed *SMVCMS* in the context of **Smart Cities** and found that the system fulfills the desired goals, such as decentralization of community management, personalized automatic management and discovery of communities, autonomy of agents and flexibility so that any agent can create its own community with the maximum level of ease.

**Keywords:** Virtual Communities, Information Exchange, *Jason*, Cartago, JaCaMo, Multi-Agent programming, Android application, Community Recommendation System

## 1. Introduction

An online Community also known as a Virtual Community (VC) is a gathering of people, in an online space where they join, communicate, and exchange information between them. Along with the time, group of people with similar interests use VCs as a synergetic way to exchange knowledge and keep in touch with their beloved ones and the world around them. VCs were originally designed to be agnostic of spatio-temporal constraints. Instead, they are nowadays spreading under *adhoc* spatio-temporal constraints in the physical environment. VCs are expanded from the paradigm to offer online spaces for users to communicate providing a digital medium for people in the same area, at the same time. The contribution goes to the advent of pervasive computing in our lives. Mobile and ambient computing lead VCs incarnation to shift from digital environments to the real world. This is the reason why there is a strong need of decentralized and open ways for managing these VCs in such settings. For example, a community within the *hospital* can serve different types of information exchanges by the patients, doctors, administration, nurses, ambulance staff, etc. Or a community server can be situated at some visiting place like *'Eiffel Tower'*, for managing the visiting people, history of the place, controlling exhibitions, reporting improvements, etc.

To tackle such requirements, we turn to multi-agent technology JaCaMo [1] Platform that showed great success in various application domains, where different autonomous decision-making entities (agents) have to communicate, exchange knowledge and cooperate in order to achieve individual and/or collective objectives [1, 2]. Using a *Multi-Agent Oriented Approach* (MAOP) for supporting VC, mobile agents act as a personal assistant on behalf of each member of a community. The agent perceives knowledge from the communities of individual interests and acts upon the communities to meet their design goals. Thus, agents bring the appropriate people having common goals or interests together and to share their knowledge with each other at a maximum ease. In addition to multi-agent platform, mobile devices adds features by building complex computational systems providing maximum level of flexibility, simplest form of interactivity, and ease of use for the end users. One such mobile technology is *Android* originated by a group of companies known as the Open Handset Alliance, led by Google. *Android*[2], as an open-source software stack for smart phones, aims at the creation of a successful real-world product that improves the mobile experiences for the end users. Recently, the launch of agent-oriented platform named *JaCa-Android* [3] provides an appropriate level of abstraction to build multi-agent based smart mobile systems and application on top of the Android platform.

We have designed and implemented a multi-agent based *Virtual Community Management System (VCMS)* based on *JaCaMo* Platform. We reported the top level architecture and demonstration of our system *VCMS* in [4]. This paper extends *VCMS* with Smart and Mobile features and introduces it as *Smart Mobile Virtual Community Management System (SMVCMS)*. In addition, this paper presents its server architecture, major functionalities such as recommendation component and also its client Android based application named as *SmartClient*. In *SMVCMS*, virtual communities are realized by means of a set of *Jason* [5] agents encapsulating the user profile and the logic and control of the specific operations involved in the community pack: *community management* (e.g. joining/leaving, creating/deleting a community), *information sharing* (e.g. publishing posts, notifying members, etc.). The agents accomplish these actions by using a set of functionalities that are made accessible by the communities they belong to. A community consists of a domain of interest, an owner, a message buffer, and an initially unspecified number of member agents, which have certain interests over the community. Agents can exploit them to achieve their individual and collective goals. These tools are implemented by means of artifacts developed in *CArtAgO* framework [6]. Agents lookup the artifacts in the different workspaces affected to each community and exploit the operations offered by each of the artifacts to achieve their desired tasks. Later in this paper we are going to address details of our system, what are the artifacts, which agents exploit them and especially the design of our mobile *SmartClient* end-user application.

As a client and server architecture, *SMVCMS* server holds the communities and clients access them from their smart Android devices installed with *SmartClient* end-user application. Since, users of *SMVCMS* vary in their interests and goals, it is nec-

[1] http://www.jacamo.sourceforge.net

[2] http://source.android.com/about/philosophy.html

essary to generate recommendations on the basis of user profiles. Recommendations by *SMVCMS* can serve as a key technology for connecting people and sharing information on the virtual world. It proposes new communities for the knowledge management and connects people having common interest with other people once having their profile on the community. *SMVCMS* lets the people updated with the information or community on the virtual space that is of their interests. Later in this paper, we are providing details how *SMVCMS* generates recommendations for its community users.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 provides a discussion on a scenario description of a smart city. Section 4 presents our Smart Mobile Virtual Community Management System. It presents *SMVCMS* artifacts and agents that perform operations on the Server and Client side. Section 5 elaborates different functionalities of our client Android system named as *SmartClient* and also presents the interface design of this application. Section 6 elaborates the use of our system in the context of smart cities. Section 7 explains different types of recommendations in our system. Section 8 presents experimental details and evaluation of the system. Finally, section 9 concludes the paper and shows our future directions.

## 2. Related Work

An adaptive room governance application is developed by A. Sorici et al. in the context of smart coworking [7]. The main contribution of their system was to focused on MAS organizations based on *Moise* framework (within *JaCaMo* platform) to establish a precise and efficient level of management for the room allocation.

A bottom-up agent-based approach for the knowledge management using virtual communities is presented by P. Maret and J. Calmet developed [8]. They developed a prototype named as *Virtual Knowledge Communities* based on the Jade system, which is a Java based software development framework that conforms to FIPA standards for intelligent agents.

An agent based *SymposiumPlanner* is developed by Z. Zhao et al. to enable topic-oriented collaboration between the distributed members of a virtual community [9]. Their *SymposiumPlanner* exploits Rule Responder that is a multi-agent system for the collaborative team and community support on the Web. A semi-autonomous rule-based personal agent assists each of the community members. Their agent infrastructure is based on the Semantic Web rules that help to capture and make cooperate for different aspects of the member's derivation and reaction logic. *SymposiumPlanner* supports the RuleML Symposia by coordinating personal agents that assist the symposium chairs, intelligently answering questions from people interested in the symposium.

A concrete agent-based architecture to proactively supply knowledge to knowledge-intensive work flow is developed by C. Toledo et al. They used *JaCaMo* platform by integrating the Business Process Management and Knowledge Management infrastructures together [10].

## 3. Smart City: A Scenario Description

*Smart Mobile Virtual Community Management system (SMVCMS)* can be used to build a *Smart Place* by employing its applied intelligence on a City, Hospital, University, Conference, Festival, Building, etc. For example, consider a *Smart City* which is equipped with community servers (*SMVCMS*) situated at different locations or buildings of the city. The purpose of community servers is to manage different communities and to provide services to its inhabitants. Communities are created on-the-fly by inhabitants of the city or by any city service. A community server is associated to a physical area defined by geographical coordinates (for instance a circle around the server). These servers can be situated at various places of the city. For instance, the following are some examples with different scenario descriptions where it can facilitate people with virtual communities.

- It can be situated in a *Theater* for the management of the community of people interested in watching the theater, interested in the history of the story behind the specific theater and certain drama or topic behind the theater, agreeing to let them be informed about the next shows matching their interests, collection of their opinions about the present and upcoming theater shows.
- In a *Fashion Show* for the management of the community of the people who participated and watched the fashion show, interested in sales and purchase of the displayed fabrics, collection of votes and choosing the best models from the participants.

- It can also be situated at any place of the city for the management of the community of citizen pushing information related to the improvement of the friendship in that place, management of the community of the citizen pushing information to the city services to signal them things to improve and repair, etc.

Wherever it is used, it facilitates groups of people for the information exchange and reuse. The community servers manage communities by offering different functionalities to its users. These functionalities are available only to users situated in a given neighborhood of the server (i.e., a certain distance or community area). A user situated in the right area can interact with the community server via his/her community assistant (or personal agent) that is installed on his/her smart device and exploit services such as create a community, join a community, publish information in the community, get information from the community, etc. These actions and community services are not feasible when the user leaves the area. Since the user may not stay in the VC for a long time, or since he/she may not be ready to interact with his/her device at that time, users delegate to the smart device the management of these different actions according to the topics that he/she is interested in. He/she expresses in advance the information he/she is ready to share a priori with other users.

## 4. Smart Mobile Virtual Community Management System

The *Smart Mobile Virtual Community Management System (SMVCMS)* that we have developed, implements a generic approach for creating "mass" of places of local exchanges and local knowledge bases in the context of smart places. The following subsections elaborate the key technology used behind its implementation, *SMVCMS* Artifacts and Agents, and different services to build and manage VCs.

### *4.1. Key Technology Used: JaCaMo Platform*

To realize the goals of *Virtual Community Management System (SMVCMS)*, we have used the *JaCaMo* platform for building the server application that serves as a central knowledge base for the communities and end users. Client application named as *SmartClient* is developed by using *JaCa-Android* infrastructure so that end-users can access different services of VC management server by their smart devices. *JaCaMo* combines three separate open source technologies, i.e., *Jason*, *CArtAgO* and *Moise* for programming MAS. Each of these programming technologies has been used for building multi-agent applications for a number of years. *Jason* has been used for programming autonomous agents. It implements the operational semantics of a variant of *AgentSpeak* and provides many user-customizable features for the development of Multi-Agent Systems [5]. *CArtAgO (Common ARTifact infrastructure for AGents Open environments)* has been used for programming environment artifacts [6]. By using this infrastructure, agents dynamically create and use artifacts as a fundamental building block for achieving their activities. Lastly, *Moise* provides infrastructure for programming multi-agent organizations [11]. By deciding to play one of the roles, according to the norms stated in the organization, agents may create a community (owner role) or participate to it (member role).With the combination of these technologies, *JaCaMo* covers all levels of abstractions that are required for the development of sophisticated multi-agent system, such as, *Virtual Community Management System*. In addition, the approach offers many advantages, such as decentralization of the community management, personalized automatic management and discovery of communities, and flexibility so that any agent can create its own community. We have investigated this approach with the idea to provide *Smart Mobile Virtual Community Management System* for smart cities.

### *4.2. Architecture of SMVCMS*

*SMVCMS* is based on a multi-agent architecture that enables community assistants (participants) to meet, share and gain quick and efficient access to the information of their interest. In *SMVCMS*, virtual communities are realized by means of a set of *Jason* agents encapsulating the user profile and the logic and control of the specific operations involved in the community pack: *community management* (e.g. joining/leaving, creating/deleting a community), *information sharing* (e.g. publishing posts), *Update Members* (notifications to members, etc.).

The agents accomplish these actions by using a set of functionalities that are made accessible by the communities they belong to. Agents can exploit them to achieve their individual and collective goals. These tools are implemented by means of artifacts developed in CArtAgO. Agents lookup the artifacts in the different workspaces affected to each commu-

nity and exploit the operations offered by each of the artifacts to achieve their desired tasks. The structure and functioning of each community is declaratively specified in an organization specification where owner, member or participant roles are defined. A community consists of a domain of interest, an owner, a message buffer, and an initially unspecified number of member agents, which have certain interests over the community. By deciding to play one of the roles, according to the norms stated in the organization, agents may create a community (owner role) or participate to it (member role). The layered architecture of *SMVCMS* is shown in Figure 1.

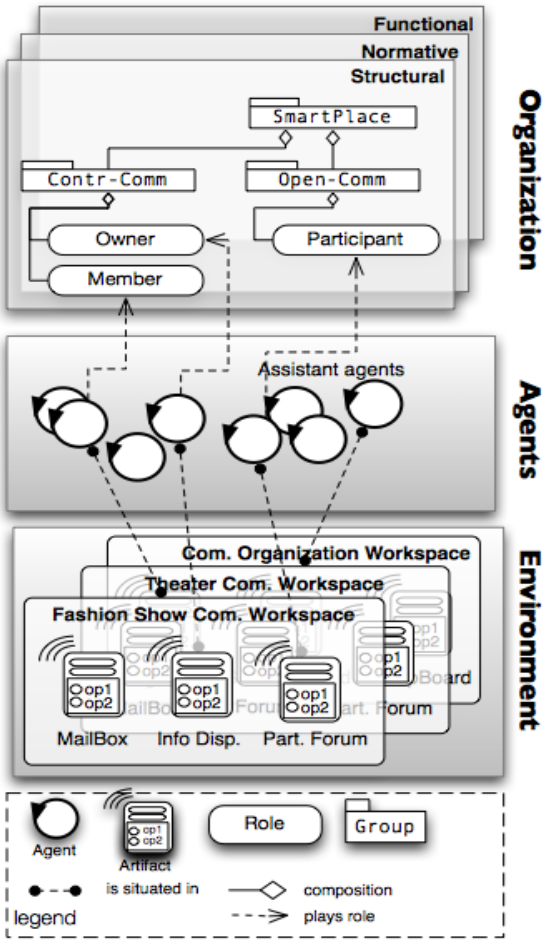



Figure 1. *SMVCMS* Architecture

*4.2.1. SMVCMS Artifacts*

The agent environment programmed as CArtAgO artifacts provides the set of basic API for agent platforms to work within *artifact-based environments*. It builds *VCs as Artifacts*, that are treated as first-class entities representing resources and tools which agents can dynamically instantiate, share and use for their desired objectives. In addition, due to its build-in capacity, it allows the synchronization and interconnection of different VCs at client and server side. The development and execution of artifact-based environment structured in open workspaces (possibly distributed across the network) where agents of different platforms belonging to several virtual communities can join and work together. The following sub-sections elaborate each of the artifacts used in the development of *SMVCMS*.

- **Community Artifact.** *'Community'* as an *'Artifact'* is the *functional-oriented* and *stateful* entity composed of controllable and observable properties. We designed a base class *"Community"* as a subclass of *"Artifact"* so that it would allow its members as a shared resource for the information exchange. Different data members of community, (such as community identifier, name, created by, list of members, list of topics, message board that captures the content of community), allow storing different types of information in *SMVCMS*. Currently, four types of communities are supported for different modes of access and exchange posts. These types of communities extend the Community Artifact class. These classes are: *{Bounded Community, UnBounded Community, Private UnBounded Community and Owned Bounded Community}*. These classes implement *'Add Message by Topic'* method according to their requirements and mode of exchange.

- **Topic KnowledgeBase (KB) Artifact**. Virtual communities compose their contents, messages, and all information under different topics. Therefore, topics of the community represent the content or posts of the community and interests of its members. Each community contains one or more Topics from *Topic Knowledge Base* (TopicKB). In addition, Topics from TopicKB belong to zero or more communities at the same time. Since, communities can overlap and share similar contents, therefore topics over the communities can be the same for several communities. Therefore, Topic Knowledge base is shared among the members so that they use existing topics to associate their messages. In this way, all the messages are managed by the Topics as it facilitates searching mechanisms. Since, Topic identifies communities messages, thus they can be created by members of the community as per their needs.

Topic are also used as search keys to lookup the messages or content of the communities. Therefore in *SMVCMS*, Topic KB is also realised as an Artifact which is shared among the users. Members of the communities share the Topic KB and associate their community to some specific Topics that exist in the *Topic KnowledgeBase*. If certain Topic is not present in KB, the member (having rights) can create it in the Topic KB and then associate it to the desired community. Since topic of the community serves the whole application (used for the recommendation, search, message organization, etc.), therefore we restrict a valid keyword having a real sense to be as a Topic. We impose *WordNet* [12] verification to avoid free texting over the communities. Each time user wants to create a new topic, system fetches all the definitions and synonyms of the user input keyword, and presents it to the user for the selection. In this way, user selects one of the senses of keyword and then system stores it in the TopicKB and associates it to the required community.

- **AgentProfile Artifact**. In *SMVCMS*, there is a need to create and maintain profiles of users that encapsulate all the required information. Therefore, we designed a class '*AgentProfile*' that represents the profile of user. This can be realized in two ways; simply a Java Object or as an Artifact. We designed *AgentProfile* as an Artifact so that it enables feature to be observed and followed by the community members of the *SMVCMS*. This is helpful knowing the status updates of the users of interest, their current locations, and activities. Being an artifact, it allows other agents to observe and communicate with other agents over the entire workspace. Therefore, profile information of each user or member of community is stored in class *AgentProfile*. It encapsulates name, origin and reputation (or history) of user as data members. In addition, each user has list of Interests, Recommendations, Notifications, Registered Communities and Owned Communities. On the basis of list of interests, mobile assistant proposes some recommendations that are stored in the list of recommendations. When the user is registered in some communities, list of Registered Communities has that information and list of notifications is updated for the user by the *SmartClient* application when any message posts are newly added to the community. When the user creates some new community, the list of Owned Community stores that particular information. Therefore, there are many methods (such as *AddProfile, AddInterest, AddCommunity*, etc.) in *AgentProfile* to update and store all types of information regarding that particular user.

#### *4.2.2. SMVCMS Agents*

In *JaCa-MAS* architecture, *Jason* agent creates, instantiates, and uses artifacts in the workspace as a fundamental building block. Therefore, we need a *Jason* agent that acts on user behalf to manipulate artifacts and manage all the client functionalities. For this reason, we need two types of agents (i.e., *Server Agent* and *Client Agent*). The server application *SMVCMS* that host communities has the '*ServerAgent*' that remains active all the time. Its main purpose is to listen for requests from the client side and act accordingly. The other client side '*ClientAgent*' serves user demands as its assistant. At first, *ClientAgent* acts on behalf of the human end-user, perceives knowledge for individual interests and acts upon them to meet ones desired goals. The information of the end-user such as name, interest, origin, current location, etc. is captured in *SMVCMS* as an Artifact *'AgentProfile'* as explained above.

Another aspect of *SMVCMS* is how client and server agents communicate and use artifacts for their objectives. It should be kept in mind that *AgentProfile* artifact stores the personal information about the community user. It is the *Jason* Agent (that acts on the behalf of the community user) which is manipulating artifacts, invoking their methods and achieving all the desired tasks. All the personal information of this *Jason* Agent is stored in *AgentProfile* artifact. The client Android application *SmartClient* has the '*ClientAgent*' that listens to the user and transmits request to the server agent and performs functionalities on the demand. Since the data on the communities may be too large and search lookup may take time, therefore there can be two possibilities to deal with the scalability issue. First, when there is a need to perform heavy operation (e.g., search), *ServerAgent* creates dynamically its *HelperAgent* as a *Jason* Agent. Then, it delivers the search or recommendation lookup tasks to the *HelperAgent* that performs the required tasks and returns results to the *ServerAgent*. In this case, community server has two *Jason* agents, i.e., *ServerAgent* and *HelperAgent*. *HelperAgent* is the assistant and is invoked by the *ServerAgent* when needed. *ServerAgent* is active all the time and listens to the client agents. But, when the number of active concurrent users is large at the same time, then server application has too much load. Instead of this technique, it is better to get arti-

facts/communities on the client device, and client agents perform their desired operations. Therefore, we implemented both the techniques to tackle the scalability issues and let community management adjust this feature according to their needs.

For facilitating interface of the Android end-user application, *ClientAgent* look-ups the internal artifacts (*connect-art, personal-art, people-art, and community-art*; explained later) to communicate with the end-user GUI (see Figure 2). *ClientAgent* exploits the operations of these local artifacts to achieve desired tasks of method invocation for user, request transmission to server, result reception and finally update of the Android interface. In this way, *ClientAgent* help their users in accessing, exchanging and managing information on the *SMVCMS*.

The *Jason ClientAgent* starts with the init goal. It looks for the local client workspace and binds it with Local Workspace ID and stores it as *belief* (i.e., informational state of the agent). *ClientAgent* needs to specify this local workspace id whenever it looks for local artifacts or performs updates on client GUI. Then, it calls the goal join remote workspace (*join_rt-ws)* with server IP and port address and binds it to the Server Workspace ID. It saves this remote workspace id as a belief and uses this id whenever it performs the operation on remote artifact or lookup artifact in remote workspace. When the join is successful, it looks for the local artifact *'connect-art'* and calls the function of the Android interface to show result for the successful connection.

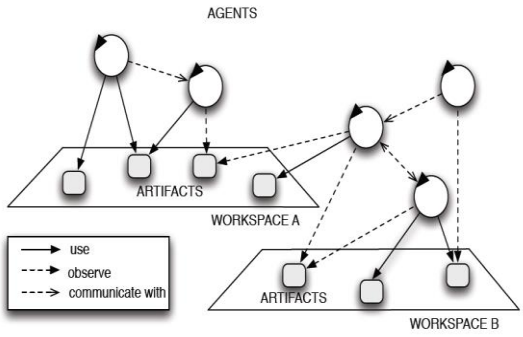



Figure 2. Agents working in different Artifacts

Figure 2 shows how *ClientAgent* can create, join and work in multiple workspaces at one time. The operations *createWorkspace*, *joinWorkspace* and *quitWorkspace* facilitate these tasks. The actions for creation and join are provided by the *NodeArtifact* and *quitWorkspace* is provided by the Workspace-Artifact. *ClientAgent* joins the local workspace by which it looks for local artifacts for the communication to human user by *GUIArtifacts*. It also joins the remote server workspace for looking different services provided by the *SMVCMS*. However, there is always a current workspace, to which all actions are routed with no artifact id or workspace id specified. Current workspace info is automatically tracked by the *current_wsp* belief. It is also possible to set the current workspace by the internal action *set_current_wsp* by providing its workspace id.

We don't go into the implementation details of these features and rest at abstract design level.

### *4.3. Community Tools for SMVCMS*

The *Smart Mobile Virtual Community Management System* allows the participants to build different kinds of communities by dynamically instrumenting their communities with different tools to manage the information shared between participants according to their requirements and needs. All the information exchanged by agents participating to a community is stored in these tools. Currently, the *SMVCMS* provides four basic classes of community tools as explained in the following subsections.

#### *4.3.1. MailBox*

This type of VC tool allows any member of the community to send and receive messages to/from other members of the community. Once an agent gets its membership to a VC, it will receive all the messages and updates posted on the community automatically. For instance, when a late incomer joins the VC, all subscriber agents receive a message about its subscription to the VC, so that they know and can initiate further communication. An illustration of this class of VC is the *'Catwalk'* on a Fashion Show, in which an organizer wants to communicate with models participating in a catwalk show, and models themselves want to communicate with each other. An ad-hoc community is therefore shaped under a publish/subscribe paradigm for information exchanges.

#### *4.3.2. Participatory Forum*

This type of VC tool allows registered members to participate in the community by different services, such as reading and writing posts on the VC. By getting registrations for the community, members will receive all the posts and updates automatically. There are many options that can be manipulated with this type of VC, i.e., *Classic, Duration, Moderation,*

*Bound*, etc. For example, the *Bounded BlackBoard* Community tool allows the creator of the VC to set the limit of the blackboard, which decides the content of the board. It lists messages on the board in an ante chronologically order. When reaching the specified limit, the buffer of the message board acts like a *First-In First-Out (FIFO)* manner and deletes the last board message to place the new message on the top of buffer. Example of such community can be *'Contest Board'* on a Fashion Show, where management displays names of models presenting the fabrics and status of designers with their votes or likings by the viewers participating in the show. The board has limited capacity and only newly important messages are placed and old messages are deleted for promoting information lookup ease.

*4.3.3. Information Dispatcher*

In this type of VC tool, only the VC owner can disseminate information. Members of the VC automatically receive messages posted on that tool. Example of such a community can be *'Catwalk Schedule'* on a Fashion Show, where only organizers can post messages about the schedule, names of selected participants, updates about displayed fabrics and brands, etc., for the registered or regular members. These members can only read the post, but they cannot post any information over the VC.

*4.3.4. Personal-Box*

In this type of VC tool, only the VC registered members can disseminate and share information over the community. Other community assistants that are non-members of the community can consult this type of tool, but they are not allowed to post messages without having membership. The owner of the community has the only right to give membership to other community assistants to get involved in the community. Example of such community can be *'Model of the Month'* on a Fashion Show community, where only registered viewers can cast a vote or judges can post their opinions for choosing the best model from the participants.

## 5. Android Client Application : *SmartClient*

The following subsections discuss architecture and modeling of our Android client application named *SmartClient*.

*5.1. SmartClient - Android Client in a Nut-Shell*

We designed an Android application, *SmartClient*, which needs to be installed on the smart phone of the user. By Android *SmartClient* application, one can access and manipulate communities on the *SMVCMS* servers. One can create community on-the-fly with different mode of exchanges and exploit other services. At the client side, *Jason* agent *ClientAgent* acts on behalf of the human user. Human user performs tasks on *GUIArtifacts* that are encapsulated by the Jaca-Activity. The services are provided by the *ServerAgent* on the server machine. *ClientAgent* communicate with the *ServerAgent* to invoke different services to fulfil desired task. Figure 3 shows the whole mechanism.

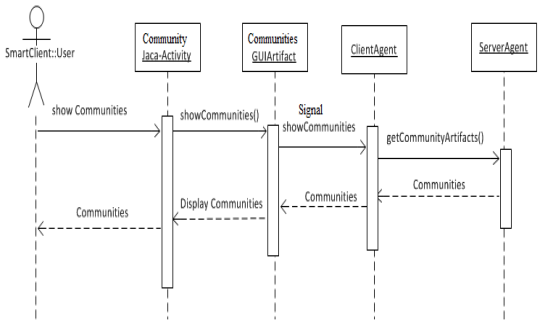



Figure 3. Interaction between *ClientAgent* and *ServerAgent*

*5.2. Local Artifacts in SmartClient Android Application*

The connectivity between the *Jason* '*ClientAgent*' and the human end-user requires to create a GUI as a subclass of *GUIArtifacts*. GUI artifact is an artifact that encapsulates IO functions that achieve interactions between human users and agents. A GUI artifact is defined by extending *GUIArtifacts*, wrapping the definition and creation of the structure of the GUI. It comprises of the components using the Swing API, and then linking/mapping Swing events into artifact's internal operations by using *link_to_primitives*. According to requirements of our project, we partition the whole functionality into four modules. These four modules are handled as *GUIArtifacts*, so that user can invoke different services in *SMVCMS*. They deal with the connectivity of client-server, personal information management, people and friend management, and community management, etc. These modules are implemented as

*Indents* in the Android application. Since operations on these GUI artifacts are managed by *Jason ClientAgent* in the Android application, therefore we create internal/local artifacts which represent *GUIArtifacts* so that *Jason* agent manipulates different service by exploiting these local artifacts. These internal artifacts *(connect-art, personal-art, people-art and community-art)* are used to invoke operations of *GUIArtifacts (Connection, Personal, People and Communities).* These local artifacts are elaborated below.

*5.2.1. Connect-Art.*

This artifact is created for managing the connectivity between the *SMVCMS* server and *SmartClient*. It interlinks the end-user with the *Jason ClientAgent* at the client side. This artifact extends the *GUIArtifacts* and is designed to invoke methods like connect and disconnect with the *SMVCMS* server. All the methods that update this GUI are implemented in *ConnectionActivity*. The *ConnectionActivity* calls the '*onCreate'* method, in which it sets the xml form *ConnectionLayout.xml* as a layout for this '*connection'* intent, and creates the *'connect-art'* local artifact of class '*ConnectionArtifact'* to communicate with *ClientAgent*.

*5.2.2. Personal-Art.*

This artifact is created for managing the personal information and activities of the end-user. Personal information includes the end-user profile, subscriptions and created communities. Therefore, this artifact deals with the services, such as creating profile, adding interest, registered and owned communities lookup, get subscription (join and leave) of communities, etc. This artifact also manages to pop-up recommendations and notifications for the end-user. For this intent, we have implemented *PersonalArtifact* that extends the *GUIArtifacts*. All the methods that update this GUI are implemented in the *PersonalActivity*. The *PersonalActivity* calls the *'onCreate'* method, in which it sets the xml form *PersonalLayout.xml* as a layout for this '*personal'* intent, and creates the '*personal-art'* local artifact of class '*PersonalArtifact*'.

*5.2.3. People-Art.*

This artifact is created for managing the people or friends for the end-user. People management includes the friend list look-up, display friendship requests, search and/or follow people. By using this, one can make a friend, send friendship request, follow a person over the community, etc. In addition, one can search member persons by name, interest or location criteria. For this intent, we implemented *PeopleArtifact* that extends the *GUIArtifacts*. All the methods that update this GUI are implemented in *PeopleActivity*. The *PeopleActivity* calls the *'onCreate'* method, in which it sets the xml form *PeopleLayout.xml* as a layout for this '*people'* intent, and creates the *'people-art'* local artifact of class '*PeopleArtifact*' to communicate with *ClientAgent*.

*5.2.4. Community-Art.*

This artifact is created for managing the communities for the end-user. Community management includes the creation and search of communities, accessing and posting information over the communities. This artifact deals with services such as create community, add topic on community, access messageboard for the information lookup, post messages on the communities, etc. In addition, one can search existing communities by the name, topic or synonym criteria. For this intent, we implemented *CommunityArtifact* that extends the *GUIArtifacts*. All the methods that update this GUI are implemented in *CommunityActivity*. The *CommunityActivity* calls the *'onCreate'* method, in which it sets the xml form *CommunityLayout.xml* as a layout for this *'communities'* intent, and creates the '*community-art'* local artifact of class '*CommunityArtifact'* to communicate with *ClientAgent*.

Finally, the main activity (named as *MainActivity*) that extends the *Jaca-Activity* encapsulates all these Jaca-Activities (*ConnectionActivity*, *PersonalActivity*, *PeopleActivity* and *CommunityActivity*) and represents them as intents (or tabs) in the Android client application. Figure 4 displays Personal and Communities intent in front. Services by these interfaces are manipulated by the *Jason* Agent '*ClientAgent*' that receives signals by the interfaces on user interaction. Then, '*ClientAgent*' can lookup these local artifacts and listen to the demands of user by the services that these intents provide. When a button (e.g., show existing communities), also called *View* in Android, is invoked by the user, then the associated operation in *GUIArtifacts* signals the *ClientAgent* that transmits the request to the *ServerAgent* on the *SMVCMS* server. *ServerAgent* does its desired functionality and returns its response (i.e., list of existing communities) to the *ClientAgent*. *ClientAgent* finally performs a lookup of the local *community-art* artifact, by which it updates the user interface with the fetched results. In this way, multi-agent architecture of *SMVCMS*,

enables community assistants (participants) to meet, share and gain quick and efficient access to information of their interest by the *SmartClient*.

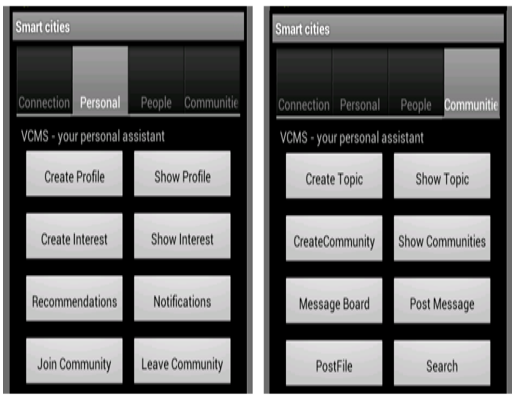



Figure 4. Personal and Communities Intents in Front in the *SmartClient* application

## 6. *SMVCMS for Smart Cities Use Case*

We have employed *SMVCMS* in the context of Smart Cities to test whether our system fulfills the desired goals. The *SMVCMS* allows participants of a city with different functionalities for building and manipulating virtual communities. A participant can create his/her own community or get the membership of existing communities. Using the options *'Join Community'* and *'Leave Community'*, one can join and leave the community if one fulfills the criteria of joining or leaving. Using the option *'Show Communities'*, one can see all the present communities with the list of topics associated with each of the community.

The *SMVCMS* allows the management of the community content within the topics or the subject of the community. Such management of community content allows flexibility in posting, searching and disseminating information within the community. With options such as *'Add Topic'* and *'Show Topic'*, participants can add a new topic in the existing community and see the existing topics under one community. It allows participants to post messages (i.e., raw content in the form of text, picture, etc.) over the community. The members are free to choose the topics of interest for the community and allow one to post the messages related to the topics over the community for the other participants. With options such as '*Add Message' and 'Show MessageBoard'*, participants can add a new message in the existing community and see the existing messages over the specific topics.

The *SMVCMS* allows participants to follow different communities of their interest. When any member joins the community, it will receive all the posts automatically in his mailbox depending on the community type and its rules. When calling the functionality '*My mailbox*' community assistant updates its owner with the list of messages about the communities in which he is registered or some other persons have mailed them. Besides this, *SMVCMS* allows participant to manipulate different informations of their interests. Once a user enters his topics of interests, his personal community assistant processes match making mechanism and explores the information over communities. Then, it brings its user with some recommendations beneficial for him. Since we reply on the profile information, It is not necessary for the user to be a member of that community. Therefore only information about his interests in his profile is sufficient to receive recommendations. With option *'My Recommendations'*, community assistant updates its owner with the list of recommendations about the things of his interest.

It is possible for participants to search for existing communities with different criteria. One can search the community by name or topic. The *SMVCMS* extracts and displays the information about the community if it exists by that name or topic. Another option for community lookup is the semantic search for the community with the help of *WordNet*. In this case, it generates the *SynSet* of the keyword entered by the user as an input and matches with the existing names of communities and/or existing topics within the communities, and then the user is prompt with the matched results if any. Besides these features, we consider semantic recommendation as a vital mechanism for our virtual community management system, hence we dedicate next section for its explanation.

## 7. Semantic Recommendations in SMVSMS

Recommendations are very important in *SMVSMS* so this section first provides an overview of research literature on generating recommendations and then elaborates how *SMVSMS* server provides recommendations to its clients.

*7.1. Recommender Literature Review*

In research literature, there are several methods, such as *content based recommendations*, *collaborative filtering*, and *semantic based methods* for generating recommendations. Several people use these methodologies in various contexts for computing recommendations. For example, news recommender system [13] recommends news items based on the user interest in a business intelligence process. *YourNews*, *NewsDude* and *PRES* systems are examples that use content based recommendation methodology. For content based recommendations, methods such as *Term Frequency-Inverse Document Frequency (TF-IDF)* and the *Cosine Similarity* measure is used. In general for producing a recommendation, content based recommendation method considers all the terms in a document. The value of TF-IDF is calculated by multiplying frequency of term and inverse document frequency. This value is computed for each term in the news items that the user has read to obtain the user profile. Cosine similarity is obtained by dot product between the number of news items with the user profile vectors, dividing by the product of the vectors magnitude. Besides content based recommendation method, semantic based method uses concepts matching strategies by exploiting Concept Equivalence, Binary Cosine and Jaccard method. Concept Equivalence matches the concepts between the two sets, news item and user profile. In binary cosine method, no. of elements of intersection set is divided by the product of no. of elements of individual set. Similarly, Jaccard method computes the similarity value by the no. of intersection set between user and item by the no. of union set between two sets. In these semantic methods, the similarity values take into account the no. of common elements between the sets, rather than frequency of words as in content based method. One of systems named as *Athena* is used for recommending news items by exploiting both methods (content and semantic based) for producing news item recommendations for the user.

Z. Yu et al. proposed ontology-based semantic recommendation system for the context-aware E-Learning [14]. It is designed for the users who want to find content that they want and need to study. The system gets user context, content of knowledge repository, and knowledge about the domain the user wishes to learn about. On this basis, the system calculates the relevance and distance to propose the learning program to the user. They calculate the semantic relevance by estimating the Conceptual proximity S between two elements (e1, e2) by these rules; (i) S(e1,e2)>0, (ii) S(e1,e2)= S(se2,e1), (iii) S(e1,e2)= Dep(e1)/M when e1=e2, (iv) S(e1,e2)= Dep(e)/M, where e is the ancestor when e1 and e2 have subsumption relation, (v) S(e1,e2)= Dep( LCA(e1,e2) )/M when e1 and e2 are different and have no subsumption relation. In these formulae, Dep is the depth of node, LCA is the Least Common Ancestor, and M is the total depth of the domain hierarchy ontology.

A. Passant proposed a music recommendation system named as *dbrec* using *DBpedia* based on Linked Data [15]. It generates recommendations for bands and solo artists by exploiting links between linked data. Their linked data semantic distance known as *LDSD* is based on four factors. These are direct and indirect links considering incoming and outcoming links in the graph to compute the distance between resources. Their recommendation system first identifies the relevant subset from *dbpedia*, reduces it for the optimization of querying, and then computes linked distance and finally provides interface for browsing recommendations.

Z. Chedrawy and S.S.R. Abidi proposed web recommender system for recommending music playlists based on the user profile [16]. They developed a hybrid approach by combining collaborative filtering with the ontology-based semantic distance measurement. First they compute an item-based collaborative filtering multi-attribute similarity by exploiting the users rating (preferences over items). Then they compute the item-based semantic similarity by counting common descriptions between two sets by dividing them with their total description count. Final value of similarity is computed by multiplying the individual similarity by their individual weight and finally adding the weighted values. Another music recommendation system is *SmartMusic* developed by W. Hu et al. for the recommendation of music lists to the user [17]. Their recommendation system analyses description of singers, songs, albums along with the user behavior and preferences.

The P. Resnik similarity measure is based on the information content of the least common subsumer (LCS) of concepts A and B [18]. Information content is a measure of the specificity of a concept, and the LCS of concepts A and B is the most specific concept that is an ancestor of both A and B. The biggest drawback of Resnik's model is that it produces the same semantic similarity between any pair of con-

cepts with the same LCS. Based on Resnik model, J. Jiang and D. Conrath's model [19] and D. Lin's model [20] takes into account edge counting as well. The C. Leacock and M. Chodorow similarity measure is based on path lengths between a pair of concepts [21]. It finds the shortest path between two concepts, and scales that value by the maximum path length found in the *IS-A* hierarchy in which they occur. A. Carbonaro and R. Ferrini propose a combined approach based on the two above measures considering both a weighted factor of the hierarchy height and a sense offsets factor [22]. T. Simpson and T. Dao adapted the somewhat same method and calculated semantic similarity as 1/Semantic Distance [23]. They computed semantic distance as 1 if same concept in the ontology. Otherwise semantic distance is the number of Archs+1; where arch is a subclass/superclass axiom between concepts in the ontology.

Y.B. Fernández et al. developed a hybrid technique for an automatic content recommendation system names as *AVATAR* based on the Semantic Web Technologies [24]. They exploited two-phase approach for the personalized TV recommendations based on semantic inferences from the ontology. In the first phase, they use the content based strategy that explores hierarchical semantic similarity and inferential semantic similarity techniques. The second phase involves the collaborative strategy. According to their experiments, the hybrid strategy significantly reduces the deficiencies of individual techniques.

D. Camacho et al. developed a MAPWEB-ETOURISM System for the tourism domain to help customers to plan their trips [25]. They designed several specialized web agents to query travel web sources and fetch individual results. Later with the help of a planning agent, individual results are integrated to build complete travel solutions. The final results serve as recommendations for the end-users to plan their trips.

### *7.2. Various types of Semantic Recommendations in SMVCMS*

*SMVCMS*, as a recommendation system for its users, is a resource to access information and connect people having common interests. Therefore, in *SMVCMS*, we designed a semantic recommendation module to propose semantic recommendations for the end-users in different scenarios and contexts. These contexts are elaborated as follows.

#### *7.2.1. Creation of a New Community*

*SMVCMS* system provides flexibility and decentralized mechanism to build communities on-the-fly by the users through their smart devices. Different people of interests get together and share information on several (may be overlapping) topics. Therefore, content on communities may overlap with each other. At the time of creation of a new community, our system gets the requirements of the new community and recommends to the user similar or overlapping communities that exist on the server repository. This type of recommendation is highly important to reduce redundant overlapping communities and control inconciseness of information over several communities. For example, consider a scenario where three music communities exist, named *{JAZZ, POP and CLASSICAL}* on the *SMVCSM*. A user wants to build a new community named *MUSIC*, *SMVCMS* system gets the requirements of this new community from the user. Then, it calculates the semantic distance between the requirement and the existing communities. Based on the semantic distance, it informs the user about the existing *JAZZ*, *POP* and *HIPHOP* communities as more specific communities that hold content related to the requirements of *MUSIC* community. In this way, the user gets awareness about the existing communities related to music and he may avoid creating another community on this topic. On viewing the existing communities, he may join the existing community and benefit from the information posted and people having common interests. However, *SMVCMS* allows one to create ones personalized communities.

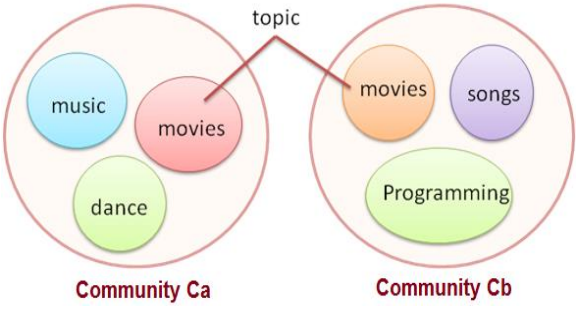



Figure 5. Semantic Recommendation based on topics between communities

In this scenario (similar topics between communi-

ties), our system calculates the semantic distance between the new community Ca and the existing communities {Cb, Cc, ..., Cn}. Let there are two communities Ca and Cb as shown in Figure 5, where Ca shows the requirements of a new community and Cb is the existing community, their semantic similarity and distance is calculated on the basis of similar topics.

*7.2.2. Information access over communities*

*SMVCMS* serves as a medium for the information access to its users. It recommends the user to join communities that contain content that matches with the interests of the user. *SMVCMS* system computes recommendation for the user which may be useful and provides awareness of existing communities on the server repository. When the user makes his profile on the *SMVCMS* server, *SMVCMS* automatically starts finding recommendations based on the interests added in the profile of the user. These recommendations are highly important for the user because they connect him/her to the right or useful information.

In this scenario (similar topic of community and interest of a user), our system calculates the semantic distances between the profile of user Pa and existing communities {Ca, Cb, ..., Cn} associated with topics of interests. Let Pa shows the profile of the user and Ca be an existing community as shown in Figure 6, semantic similarity and distance between them is calculated on the basis of similarity between the topics of community and interests of person in his profile.

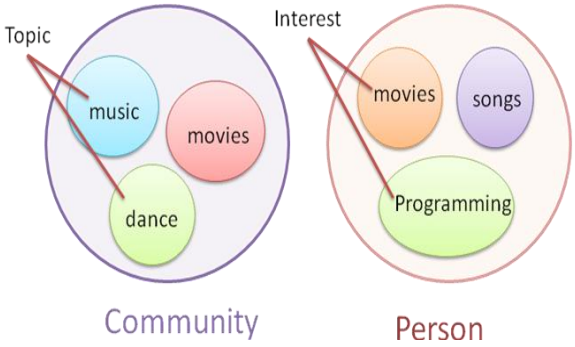



Figure 6. Semantic Recommendation based on topic of a community and interest of a person

*7.2.3. Connecting People based on interests*

*SMVCMS* serves as a tool for connecting people sharing similar interests. It recommends possible users as friends having similar interests. For this purpose, it has to compute the semantic distances between the profiles of the users and proposes friends for the members of communities over the server. *SMVCMS* system computes recommendation for the user which may be useful, providing awareness of existing people utilizing *SMVCMS*. When the user makes his profile on the *SMVCMS* server, *SMVCMS* automatically starts finding recommendations based on the interests added in the profile of the user. These recommendations are highly important for the user and they connect the user to the right people having common interests.

In this scenario (similar interests between users), our system calculates the semantic distances between the profile of user Pa and existing profiles {Pb, Pc, ..., Pn}. Let Pa be the profile of the user and Pb be the existing profile as shown in Figure 7, semantic similarity and distance between them is calculated on the basis of their common interests.

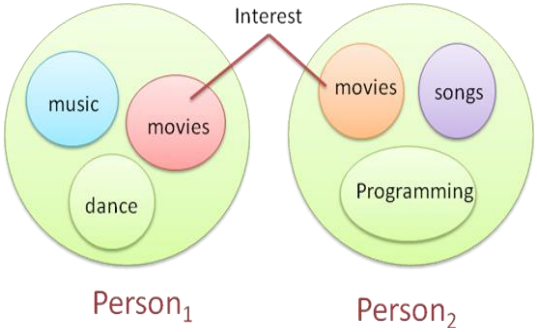



Figure 7. Semantic distance based on interests between persons

*7.2.4. Search for Community or People*

*SMVCMS* provides users an interface for searching community or people with keywords. Former is similar to the section information access over communities, and later one with connecting people based on interests. The difference lies in the execution in the sense that user has to enter keyword and manually trigger the search, but in the previous case, once a user has made his profile *SMVCMS* automatically executes the semantic distance to recommend the user different communities that may be appealing to the user based on the interests in his profile.

*7.3. Semantic Distance calculations in SMVCMS*

In *SMVCMS*, we compute semantic distance and use it for generating recommendations for the user. Once a user builds his profile, the recommendation system activates to guide the user about the communities and people of his interest. Our proposal for semantic similarity and semantic distance calculation is based on the hybrid technique. For recommendations, we need to match interest of user and topics of community. We restrict the user to select the topic of community from the domain ontology at the time of community creation. We also restrict user to add interest in his profile as a concept from the domain ontology. Therefore, the problem reduces to calculate the semantic distance between the concepts of domain ontology, one belonging to community and other as a user interest. For this purpose, we explore the explicit and implicit relations between the concepts within the domain ontology. Then we apply the semantic method (Jaccard similarity) to compute the semantic similarity between concepts. We also aim to compare the hybrid technique with the individual content based recommendation techniques.

*7.3.1. Analysis by Subsumption Relations*

First, we analyze hierarchical links by *Subsumption* relations. In this regard, the method proposed by Yu et al. and T. Simpson is helpful. But with in-depth analysis, both these methods show that they have limitations. In Z. Yu et al. [14] semantic relevance appears the same when the concepts are at different locations, but, having the same ancestor. For instance, let depth of ontology hierarchy be 8, and Cb and Cx are two concepts which are children of Ca located at depth 2 in the ontology. Concepts (Ca, Cb) have path length by subclass of axiom 2 and concepts(Ca, Cx) with path length 7. The semantic relevance between them is the same between these two cases (2/8 = 0.25) inspite-of their distance between them.

The T. Simpson method [23], which is actually adapted from Resnik's similarity measure [18], has a limitation that it produces the same semantic similarity between any pair of concepts with the same *Least Common Subsumer*. However, we employ these methods because they are simple in implementation and helpful in computing the semantic proximity or hierarchal differences between concepts.

*7.3.2. Analysis by Property Relations*

Second, we analyze associations by property links between the concepts. Datatype and Object properties in the ontology represent the context and semantics of concepts. Generally, datatype properties are called the attributes of a concept in the ontology. For example, each Book concept has some attributes such as ISBN, Name, Price, etc. Object properties or relations make direct and reciprocal links between concepts within an ontology. For example, Object properties *Contributes(Author, Paper)* and *isReviewedBy( Paper, PcMembers)* make associations between the concepts and represent the real descriptions, which help to evaluate the semantic similarity between them. This is closely similar to the inferential semantic similarity methodology proposed by Y.B. Fernández et al. [24], and is helpful in finding the semantic similarity more precisely than with only hierarchical information by analyzing the relations between concepts in the ontology.

*7.3.3. Analysis by Implicit Relations*

Third, we explore *Implicit Relations* between Concepts. It is of high importance to determine the implicit relations between concepts. Although this aspect is not much explored in the context of recommendations, but, in the semantic search this idea is well elaborated for finding the associations between concepts. Therefore, we need to explore all types of implicit information between concepts through subsumption, properties, and instance relations in the ontology. Generally, concepts are linked together by object properties (denoted by arcs) and labeled with the property name. According to B. Aleman-Meza et al. [26] semantic associations between concepts are based on notions such as connectivity and semantic similarity. Two classes are semantically associated if they are semantically connected or semantically similar. Semantically connectivity means that there exists a path between them by properties and intermediate concepts. Semantic similarity relates two classes that do not have a direct link but they are related by a sequence of interconnected links, subsumption of concepts and properties, instances, etc.

When the concepts are semantically associated, then they are more semantically similar and their distance is smaller. We hope this addition in the semantic distance calculation helps in achieving more accurate results, then only using hierarchical or direct links (subsumptions or properties) between concepts.

We have implemented these three methods and finally get their aggregative result for the semantic distance in our application. Based on the aggregative result, our recommendation system detects three kinds of recommendations, exact match, near-close and more specific for the user based on interests provided in the profile. In general, the recommendation system for proposing communities to the user having interests works as follows. We get the *synonyms, hypernyms* and *hyponyms* of the concept which represents users' interest. Then, it locates in the topic ontology. If found, then the system has a recommendation to propose. Then, it searches the neighborhood to propose some more recommendations. In the case of no community under that topic, the system gets hyponyms of the concept (i.e., interest) and proposes more general recommendations. In the case of too many communities under that topic, the system gets hypernyms of interest concept and proposes more specific recommendations. Concepts which are far by hierarchical relations may be very close to each other by property relation. The more the ontology is enriched with relations between concepts, the more the hybrid approach is able to find associations (direct and indirect) between concepts which results in better recommendations.

We conclude that explicit property relations and implicit inferences play an important role in generating recommendations. Only hierarchical relations lookup for generating recommendations is not enough. *At the time of writing this paper*, our recommendation system is not well advanced. Therefore, we are unable to present empirical evaluation of our semantic model. Hopefully, we will include the work in this direction in our ongoing research.

## 8. Experimental Results

We have tested our system in a real-life scenario at *Château de Versailles* (gardens at Paris) in context of *Smart City*. *MSVCMS* servers are installed at various locations in the gardens of Versailles. The following sub-sections discuss various aspects of our experiment in detail.

### *8.1. Background of Test Scenario*

- **Various Communities**. We have created various communities under several policies with different objectives to test the VCMS in the real environment at Versailles. At Château de Versailles following communities are running before the evaluation of a scenario: *{'Hall of Mirrors, Grand Versailles', 'Jardins à la française', 'Information Desk', 'Report Suggestion, Leave Comment', 'Sale and Purchase Items', 'Versailles - Chinese Friend', 'Friend of Versailles', 'Explore Musee', 'Lesson of the day', etc. }.*
- **New Users for test**. To test a scenario, we incorporated three persons, (Alice, Jim and Bob) who are friends. Alice is an expert of cultural exchanges between France and China during the XVII Century. She has some information (dates, facts, texts) stored on her smart device. Jim is the student of Music and interested to meet people having interests in music. Bob lives in Paris. Alice and Jim are coming from China to Paris for participating in an international conference. In actual, coming to Paris is a realization of their dreams. They are excited for participating in the conference and also for exploring the beauty of Paris. But, they have only three days and in this limited time they asked their friend Bob to help them. Unfortunately, when they arrived, Bob told them about his busy schedule. But, he told them that Paris is a *'Smart City'* and they can access all the information from their *'Smart Devices'* once having *'Smart Community Assistant'*. He wished to join them in Versailles later-on.
- **Regular users of Communities**. Before testing a scenario, three persons (John, David and Nico) are already using the communities as regular users of communities for achieving their purposes. John is a guide for the visitors at Versailles. He regularly posts his schedule for the guided tours on '*Jardins à la française*' community. David is a music composer at Versailles. He delivers music to the visitors and arranges music lessons for the interested participants. David posts his notes on the *'Lesson of the day'* community. Nico receives notes by David and is a regular member of this community.

### *8.2. Evaluation Detail on the Test Scenario*

This section presents two test scenarios to show the usability of our system in the smart city environment and how it helps new users to profit from virtual communities. The detail is as follows:
Alice and Jim arrive at *Château de Versailles*. After reaching they install *Smart Community Assistant (SmartClient)* on their smart devices to connect with the *MSVCMS* server.

*8.2.1. Test Scenario 1.*

1. Alice builds her profile. She enters topics of her interest: {*garden, cultural exchanges*}.
2. Her personal Assistant (SmartClient) proposes her to join communities *'Jardins à la française, Grand Versailles*' on the basis of her interests. Given her nationality, *SmartClient* also proposes her to join the "*Versailles Chinese Friend*" community. She joins these communities.
3. Her personal assistant (i.e., *SmartClient)* regularly scans information exchanges and members of the community. It identifies (displays, stores) relevant information to Alice: *{maps, path to see flowers, etc.}*
4. Her personal assistant notifies her about the '*Guided Trip*' with the *Guide John* who explains the history of places and responds to the visitors' questions. She joins her Guided tour on time, thanks to her personal assistant which dispatches schedule information beforehand.
5. Since Alice was excited about '*Cultural exchanges between France and China*', and since no community existed on this topic, she creates an open exchange community on this topic. She enters community name '*Cultural exchanges between France and China'*, and creates topics *{'culture', 'heritage'}* for her community.
6. She selects some data and files to post a message on her new community created by herself. She selects file names *'culture.txt', 'chinese special moments.txt'*, and many other files from her smart device. The information and files data are available on her community for the other community members who will join that community.
7. Her Guide, John, creates a community named '*Explore Versailles with John*' to send his schedule to the visitors, (John chooses *'InformationDispatcher'* as a community type, as he only wants to send notification about his schedule and his explanations). He enters community name *'Explore Versailles with John'* and creates topics *{'Schedule', 'Agenda'}* on his community.
8. Alice's personal assistant delivers information related to Members of the Versailles royal court visiting China.
9. The personal assistant of another visitor joins the new community created by Alice about cultural exchanges.
10. The personal assistant of the other visitor receives this information posted by Alice.

*8.2.2. Test Scenario 2.*

1. Jim builds his profiles. Jim enters topics of his interest: {*music, architecture, friendship*}.
2. Jim's personal Assistant (i.e., *SmartClient)* proposes him to join *{'Hall of Mirrors, Grand Versailles'}* on the basis of his interests *architecture*. It proposes him to make friends {'David', 'Nico'} on the basis of his interest in *music* and *friendship*.
3. He joins this community. Jim also decided to make friendship with David and Nico, so he sends them a friendship request.
4. Jim's personal Assistant notifies him that David is his friend now. It also notifies Jim that his friendship request is sent to Nico. (As David is open to make friends so Jim receives acceptance notification as soon as he sent him friendship request as per his profile settings. But, Nico has a restricted profile so first he will see the other person's profile or meet him, only then decides to accept/reject friendship request).
5. Jim sends a message to David and by then they exchanged some messages. Jim also starts following David.
6. David updates his status, "*Going to teach Music Class"*.
7. Jim's assistant notifies him about David's status and which makes Jim excited to meet and share Music themes. Therefore, he contacts David who proposed him to join his Music class.
8. Jim joins Music class and became happy to meet David and get a lesson. He also meets Nico (Jim's country mate) in the class who is studying music from David.
9. Jim also joins David's Music Notes community, David allows him after his registration to the community.
10. Jim request to gets David's Music Notes community on his smart phone, as he wants to learn notes when he will leave Versailles. David allowed him to get his Music Notes community.
11. After music class, Nico wishes to meet Alice. Nico is very happy to meet his country fellows.
12. In the evening, Bob also arrives at Versailles. As a regular user, he has already personal assistant on his smart device. He just joins community '*Versailles - Chinese Friend'*. Bob sees the list of community members, where he tracks his friends online by message exchanges. Three friends with their new friend at Versailles meet each other and get together.

*8.3. Functionalities demonstrated by the Scenarios*

- Personal assistant activates automatically when someone builds his profile with interests.
- Personal assistant brings recommendations about communities on the basis of one's interests. It also brings suggestions about friends on the basis of one's interests.
- One can get membership of a community if one meets criteria. Similarly, one can make friend or follow person from community users if one meets criteria.
- After getting membership, it is possible to see information exchanges and members of community, share new information; start a new topic, etc. It is also possible to see existing communities, and join them if criteria meet.
- Registered members of community get notification messages over the community. Following person gets notification messages when the follower changes his status.
- One can create community any time with different mode of exchanges. Similarly, one gets information exchange and notifications of new information by one's personal assistant.
- Registered members of community can exchange information with each other.
- One can find communities or persons (users) by the different criteria.

With the use of *SMVCMS* server application and *SmartClient* personal assistant application, we showed that Alice and Jim have a convenient way to exchange (get and send) relevant information and meet friends during their visit in the gardens of *'Chateau de Versailles'*. With the demonstration of *SMVCMS* and *SmartClient* in a real environment, we conclude that multi-agent based virtual community management system is quite helpful in building complex software systems. We believe that our development is a milestone towards building a smart place, such as a smart city, for linking people and knowledge exchange.

## 9. Conclusion and Future Work

The Virtual Communities are synergetic digital spaces to build collaborations and exchange knowledge between people sharing similar interests or goals. They respond to human needs such as information sharing, friendship, and recreation. Multi-Agent Oriented Programming approach facilitates VC Management by providing autonomous features that are required in today's mobile environments.

In this paper, we present *Smart Mobile Virtual Community Management System* that is built by using the *JaCaMo* platform, where agents as personal assistants can meet and share knowledge with other agents who share a similar domain of interest. It presents the global architecture of the *SMVCMS* server application by providing an overview on how autonomous *Jason* agent treats communities as artifacts and model environment as first class entities to achieve their goals. It presents *SMVCMS* artifacts, their observable and controllable properties and operations, and customizable approach to the creation of different communities with different possible functions and modes of exchanges. Finally, it presents how *SmartClient* application is built on the top of Android platform so that end-users can access and gain from *SMVCMS* by their smart devices. In addition, *SMVCMS* exploits semantics to propose recommendations for end users. Its semantic model for the generation of recommendations employs a hybrid technique for providing recommendation on-the-fly. Initial results show that a hybrid technique shows better results than by only hierarchical analysis. It implements a customizable approach to the creation of different communities, with different possible functions and modes of exchanges. It enables VCs to be a place where agents can meet and exchange knowledge with other agents who share a similar domain of interest.

We find *JaCaMo* as a power technology for building complex multi agent system encapsulating three different technologies. Especially, the agent environment programming as *CArtAgO* artifacts provides the set of basic API for agent platforms to work within artifact-based environments. It builds VCs as artifacts, that are treated as first-class entities representing resources and tools that agents can dynamically instantiate, share and use for their desired objectives. In addition, due to its built-in capacity, it allows the interconnection of different VCs. The development and execution of artifact-based environment structured in open workspaces (possibly distributed across the network) where agents of different platforms belonging to several virtual communities can join and work together.

Using the MAOP *JaCaMo* platform, it was possible to install a decentralized and open management of those communities in a context of smart cities. The development and use of *SMVCMS* in the context of smart city, where inhabitant of the city access com-

munity servers installed at various points of the cities for information exchange and connecting people, is highly useful. In addition, the approach offers many advantages, such as decentralization of the community management, personalized automatic management and discovery of communities, and flexibility so that any agent can create its own community on-the-fly.

Our ongoing research on this topic is to test and present the empirical evaluation and results of the recommendation module of our *SMVCMS*. Our future direction is to handle the organization of different communities and authorizations to agents with *Moise* framework using *JaCaMo* platform.

## 10. ACKNOWLEDGMENTS

This work has been partially supported by the Conseil Général de la Loire, France.

## References

[1] N. R. Jennings. An agent-based approach for building complex software systems. Commun. ACM, 44(4):35-41, 01.

[2] Öztürk, Pinar, Kari Rossland, and Odd Erik Gundersen. A multiagent framework for coordinated parallel problem solving. Applied Intelligence 33, no. 2 (2010): 132-143.

[3] A. Santi, M. Guidi, A. Ricci, JaCa-Android: An Agent-based Platform for Building Smart Mobile Applications, in: M. Dastani, A. El Fallah Seghrouchni, J. H□ubner, J. Leite (Eds.), Languages, Methodologies, and Development Tools for Multi-Agent Systems, vol. 6822 of LNAI, Springer, pp. 95-119, 2011.

[4] M. Fahad, O. Boissier, P. Maret, C. Gravier Smart places: multi-agent based virtual community management system, WI&C '12 held with the 21st WWW 2012, pp. 2, Lyon, France, 2012

[5] R. Bordini, J. Hübner, and M. Wooldridge. Programming Multi-Agent Systems in AgentSpeak Using *Jason*. John Wiley & Sons, Ltd, 2007.

[6] A. Ricci, M. Piunti, M. Viroli, and A. Omicini. Environment programming in CArtAgO. Multi-Agent Programming: Languages, Platforms and Applications, Vol. 2. springer 2009.

[7] A. Sorici, O. Boissier, G. Picard, A. Santi, Exploiting the JaCaMo Framework for Realising an Adaptive Room Governance Application, ACM Workshop (AGERE'11)

[8] P. Maret and J. Calmet, Agent-based knowledge communities, International Journal of Computer Science and Applications, Vol. 6, No. 2, pp 1-18, 2009

[9] Z. Zhao, A. Paschke, C.U. Ali, and H. Boley, Principles of the SymposiumPlanner Instantiations of Rule Responder, RuleML'11, LNCS 7018, pp. 97-111, 2011

[10] C. Toledo, R. H. Bordini, O. Chiotti, and M. R. Galli, Developing a Knowledge Management Multi-Agent System Using JaCaMo, Workshop ProMAS, AAMAS 2011

[11] J. F. Hübner, J. S. Sichman, and O. Boissier. Developing Organised Multi-Agent Systems Using the MOISE+ Model: Programming Issues at the System and Agent Levels. Agent-Oriented Software Engineering, 1(3/4): pp. 370–395, 2007.

[12] C. Fellbaum, WordNet: An Electronic Lexical Database. Cambridge, MA: MIT Press. 1998

[13] F. Frasincar, W. IJntema, F. Goossen, F. Hogenboom, A Semantic Approach for News Recommendation. Business Intelligence Applications and the Web: Models, Systems and Technologies (2011): pp. 102.

[14] Z. Yu, Y. Nakamura, S. Jang, S. Kajita, and K. Mase, Ontology-Based Semantic Recommendation for Context-Aware E-Learning. Ubiquitous Intelligence and Computing, Lecture Notes in Computer Science Volume 4611, 2007, pp 898-907

[15] A. Passant, Dbrec - Music Recommendations Using DBpedia. "Dbrec - music recommendations using DBpedia." In The Semantic Web–ISWC 2010, pp. 209-224. Springer Berlin Heidelberg, 2010.

[16] Z. Chedrawy and S.S.R. Abidi, A Web Recommender System for Recommending, Predicting and Personalizing Music Playlists. Web Information Systems Engineering - WISE 2009 Lecture Notes in Computer Science Volume 5802, 2009, pp 335-342

[17] W. Hu, K. Yan, C. Jia, and J. Wu, SmartMusic: An Online Music Recommendation System Based on Semantic Web Technology. Semantic Web Challenge held at The 10th International Semantic Web Conference, 2011

[18] P. Resnik, (1995). Using information content to evaluate semantic similarity in a taxonomy. Proceeding IJCAI'95 Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1, pp. 448-453

[19] J. Jiang, and D. Conrath (1997). Semantic similarity based on corpus statistics and lexical taxonomy. Proceedings on international conference on research in computational linguistics, Taiwan. CoRR cmp-lg/9709008 (1997)

[20] D. Lin, (1997). Using syntactic dependency as a local context to resolve word sense ambiguity. Proceedings of the 35th annual meeting of the association for computational linguistics, Madrid. pp. 64-71

[21] C. Leacock, and M. Chodorow (1998). Combining local context and WordNet similarity for word sense identification. WordNet: An electronic lexical database. C. Fellbaum, MIT Press: pp. 265-283.

[22] A. Carbonaro and R. Ferrini, Concepts-based Content Analysis for Semantic Recommendations. In ECAI 2006 Workshop on Recommender Systems, pp. 57. 2006.

[23] T. Simpson and T. Dao, Wordnet-based semantic similarity measurement.codeproject.com/cs/library/semanticsimilarity wordnet.asp , 8 Feb 2010

[24] Y. B. Fernández, J. J. P. Arias, M. L. Nores, A. G. Solla, and M. R. Cabrer. AVATAR: An improved solution for personalized TV based on semantic inference. IEEE Transactions on Consumer Electronics, (2006), vol. 52(1), pp.223-231.

[25] D. Camacho, A. Ricardo, D. Borrajo, and J. M. Molina. Multi-agent plan based information gathering. Applied Intelligence 25, no. 1 (2006) : 59-71.

[26] B. Aleman-Meza, C. Halaschek, I.B. Arpinar, and A. Sheth, Context-Aware Semantic Association Ranking. In SWDB, vol. 3, pp. 33-50. 2003.