Project no.:
AAL-2009-2-137

PeerAssist

A P2P platform supporting virtual communities to assist independent living of senior citizens

Deliverable 5.2
"PeerAssist Semantic Layer"

Lead Participant/Editor	seekda/Emilia Cimpian
Authors	Michael Fried (STI), Emilia Cimpian (seekda)

Table of Contents

1 Introduction

This report focuses on the implementation of the Semantic Layer Agent (SLA) of the PeerAssist system. Work package 5 concentrates on the implementation of the overall system, thus making this the applied contribution of the project. The SLA component performs semantic matching and persistently stores data within an RDF enabled repository.

First, we shortly reflect on the specification agreed upon in the previous work packages. Herein, we revisit semantic matching systems. The next section describes the software in detail. Here we focus on the API as well as the persistency and querying mechanism. Before concluding, we will describe how to install and configure the Semantic Layer Agent component.

2 Reflection on the specification

This section recaps the specification as described in work package 3 and 4 deliverables and explains which parts were implemented in the PeerAssist prototype.

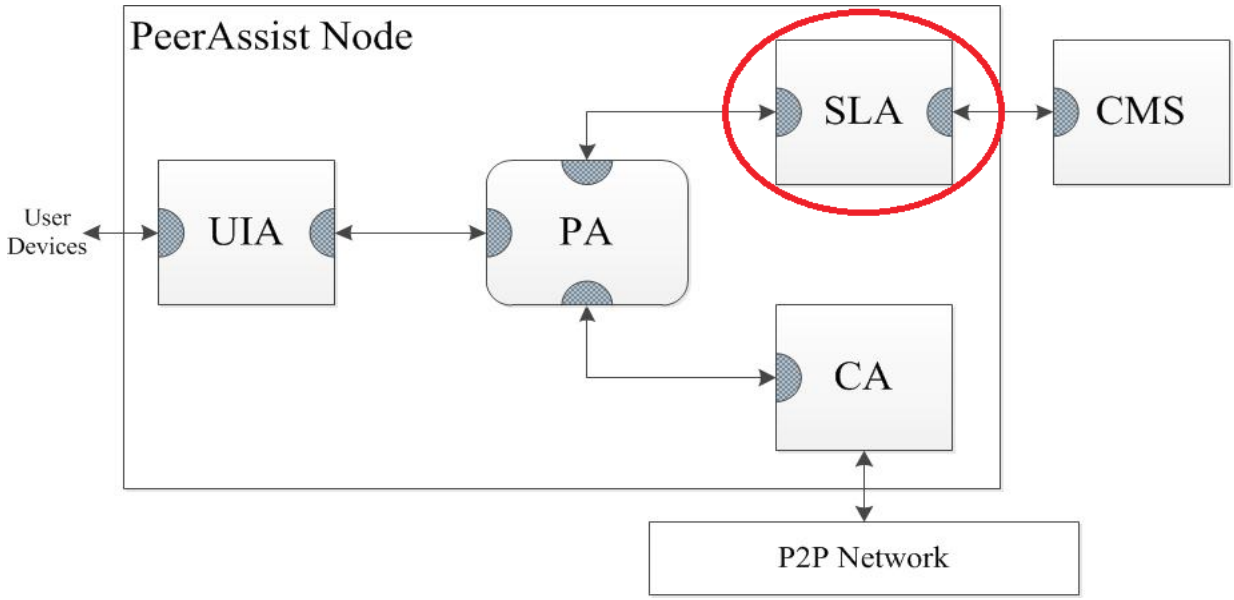

Figure 1: High level component overview

The SLA resides within each PeerAssist node. It is accessed solely by the Personal Assistant and redirects queries to the Central Matching System.

As specified, the SLA maintains user profile and local context information in the Local Semantic Information (LSI) knowledge base and interacts with the Central Matching System (CMS) when needed to process user queries. Also it retrieves information about other peers within the system from the CMS.

In order to guarantee persistent storage and synchronization of data, hence providing a smooth user experience during the trials, we ultimately decided to scrap the LSI. All information is stored on the CMS and accessed from there. However the LSI capabilities are already prepared in the prototype but remain unused.

In order to satisfy the needs of all imported ontologies we decided to use the OWL dialect for the PeerAssist general ontology. As semantic repository we use the sesame solution. The advantage is that the CMS can be easily realized via a sesame remote repository.

3 Software description

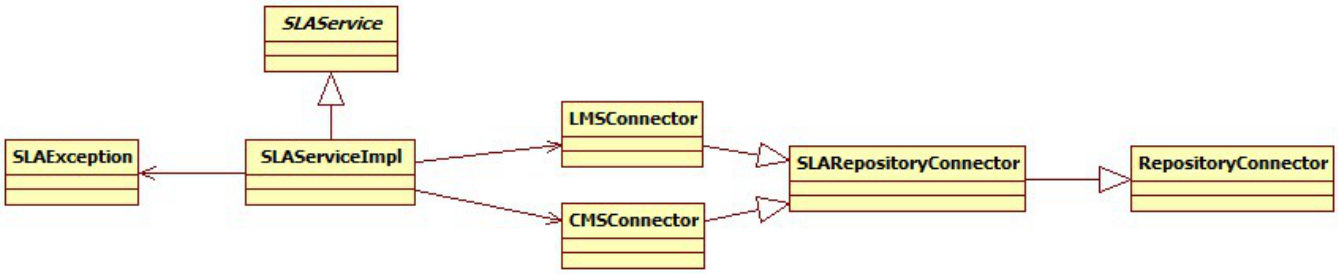

Figure 2: SLA classes overview

In order to achieve an RDF based persistent storage for PeerAssist multiple classes were introduced. The system works as follows:

- SLAService
  - Main interface which exposes the SLA functionality to the Personal Assistant. It contains all methods which can be accessed from the outside.
- SLAServiceImpl
  - Implements the SLAService interface. This class decides which information is stored on the local or global matching system. It also handles transaction management and rollback actions in case communication with the CMS fails.
- SLAException
  - Is thrown if something goes wrong in the SLA. It also wraps exceptions coming from lower layers and relays them to the Personal Assistant component.
- RepositoryConnector

- Handles the raw extraction of RDF data as well as implements the actual rollback, commit and close methods performed on the repository.

- SLARepositoryConnector
  - Inherits from RepositoryConnector and performs the actual storage. Also roughly covers the methods of the SLAService interface.

- CMS/LMSConnector
  - Both inherit SLARepository Connector. If data storage differs locally and globally e.g. some parts of data are only available locally the methods of SLARepositoryConnector have to be overridden. This means storage can be implemented separately for the local and global matching system in order to decide which kind of data goes where.

3.1 Semantic Layer API

At the core of the SLA the SLAService exposes all the functionality to the Personal Assistant which handles the communication between the User Interface Agent and SLA persistency component. In this Section we describe the methods and their respective return values.

Figure 3: SLAService methods

- String createActivity(Activity activity)
  - Creates a new activity. An activity within the ontology is identified via a full URI looking like:

    http://cnl.di.uoa.gr/peerassist/general#activities/activityId

    Within the rest of the system only the last part, the activityId, of the URL is passed around and the activity data type only contains this last part. This activityId and the creator's userId are stored in the ontology.
  - Returns the activityId formed by username_a_UUID.
- String createBlogPost(String groupId, BlogPost post)
  - Creates a new blog post. A blog post within the ontology is identified via a full URI looking like:

    http://cnl.di.uoa.gr/peerassist/general#posts/postId

    Within the rest of the system only the last part, the postId, of the URL is passed around and the blog post data type only contains this last part. This postId and the creator's userId, are stored in the ontology.
  - Returns the postId formed by username_p_UUI.
- String createEvent(Event event)
  - Creates a new event. An event within the ontology is identified via a full URI looking like:

    http://cnl.di.uoa.gr/peerassist/general#events/eventId

    Within the rest of the system only the last part, the eventId, of the URL is passed around and the event data type only contains this last part. This eventId and the creator's userId are stored in the ontology.
  - Returns eventId formed by username_e_UUID.
- String createGroup(Group group)
  - Creates a new group. A group within the ontology is identified via a full URI looking like:

    http://cnl.di.uoa.gr/peerassist/general#groups/groupId

    Within the rest of the system only the last part, the groupId, of the URL is passed around and the data type only contains this last part. This groupId and the creator's userId are stored in the ontology. The creator is automatically added to the group via a group join.
  - Returns groupId formed by username_g_UUID.

- String createNotification(String originator, String receiver, NotificationType notificationType, Long timestamp, String groupId, String message)
    - Creates a new notification. A notification within the ontology is identified via a full URI looking like:

      http://cnl.di.uoa.gr/peerassist/general#notifications/notificationId

      Within the rest of the system only the last part, the notificationId, of the URL is passed around and the data type only contains this last part. This notificationId and the originator, the userId, are stored in the ontology.
    - Returns notificationId formed by username_n_UUID.
- String createUser(User user)
    - Creates a new user. Note that the username equals the userId. A user within the ontology is identified via a full URI looking like:

      http://cnl.di.uoa.gr/peerassist/general#users/username

      Within the rest of the system only the last part, the username, of the URL is passed around and the user data type only contains this last part. This method does not return a new userId because the username must be known and set prior to calling this method. The medical profile is created automatically with the following URI:

      http://cnl.di.uoa.gr/peerassist/general#medicalProfiles/userId
    - Returns userid when creating was successful. At the moment just the same userid, which equals the username, as in the user object is returned.
- String createService(Service service)
    - Creates a new service. A service within the ontology is identified via a full URI looking like:

      http://cnl.di.uoa.gr/peerassist/general#services/serviceId

      Within the rest of the system only the last part, the serivceId, of the URL is passed around and the data type only contains this last part. This serviceId and the creator's userId are stored in the ontology.
    - Returns serviceId formed by username_s_UUID.
- void deleteActivity(String activityId)
    - Deletes an activity from the system. Only the activity creator can delete the activity.
- void deleteBlogPost(String postId)

- Deletes a blog post from the system. Only the creator can delete the post.

- void deleteEvent(String eventId)
  - Deletes an event from the system. Only the event creator can delete the event.
- void deleteGroup(String groupId)
  - Deletes a group from the system. Only the group creator can delete the group.
- void deleteService(String serviceId)
  - Deletes a service from the system. Only the service creator can delete the service.
- void deleteUser(String userId)
  - Deletes a user from the system. Also the medical profile will be deleted.
- Activity getActivity(String activityId)
  - Returns an activity object for a given activityId.
- BlogPost getBlogPost(String postId)
  - Returns a blog post object for a given postId.
- Collection<BlogPost> getBlogPosts(String groupId, String userId, int limit, int offset)
  - Returns the posts that belong to the sioc:Container which is linked to the given groupId (if not null), or to userId (else). If groupId and userId are not null, also the results are filtered by creator. Returned results are sorted by date most recent first.
  - Returns a collection of blog posts.
- Event getEvent(String eventId)
  - Returns an event object for a given eventId.
- Group getGroup(String groupId)
  - Returns a group object for a given groupId.
- Collection<User> getGroupMembers(String groupId, int limit, int offset)
  - Returns a Collection of Users a list of all group members as full user objects or an empty list if group does not exist.
- Collection<User> getMatchingUsersWithCountry(String groupId, String country, int limit, int offset)
  - Returns a list of users that match a certain group profile.

- MedicalProfile getMedicalProfile(String userId)
  - Returns a medical profile for a given userId. The medical profile URI is generated automatically when creating a user.
- getMemberState(String userId, String groupId)
  - Returns the member state regarding a group.
- Notification getNotification(String notificationId)
  - Get a specific notification object.
- Collection<Notification> getNotifications(NotificationType type, Boolean read, int limit, int offset)
  - Get notification objects according to filters. If a filter is null it will be ignored in the query. The results are sorted by date, most recent first. Only notifications are returned which have the current user as receiver.
- Collection<Notification> getNotifications(String originator, String receiver, String groupId, NotificationType type, Boolean read, int limit, int offset)
  - Get notification objects according to filters. If a filter is null it will be ignored in the query. The results are sorted by date, most recent first. Only notifications are returned which have the current user as receiver.
- Service getService(String serviceId)
  - Returns a service object for a given serviceId.
- Collection<Object> getSuggestions(SuggestionsType suggestionsType)
  - Returns suggestions for the specific user. Objects should be adjusted accordingly. Suggestion types include: Group, Activity, Event and User. Users will be recommended based upon the number of matching topics of interest between a user and another user. Groups will be recommended based upon the number of matching topics of interest between a group and a user. Activities will be recommended based upon the number of matching topics of interest between the activity's group and a user. Events will be recommended based upon the number of matching topics of interest between the event's group and a user.

    If no matching topics exist searched topics, ordered by search count, will be compared to the group's or user's topic of interest.
- User getUser(String userId)
  - Get a user's info. If it is the same user, the full user profile is returned, otherwise the public profile is returned (i.e., null values in private fields).

- Collection<Group> getUserGroups(String userId, int limit, int offset)
  - Returns all groups which a user is member of.
- Collection<QueryField> getQueryFields(String queryId)
  - Get the list of QueryFields associated with the query identified by queryId.
- Collection<String> getSuggestedValues(String queryId, String field)
  - Get a list of suggested values for the given field and query.
- void inviteUsers(String groupId, String[] userList, String message)
  - Invite multiple users to join a private group.
- boolean isInvited(String userId, String groupId)
  - Check if user is invited to a group.
- boolean isMember(String userId, String groupId)
  - Check if user is member of a group.
- void joinGroup(String userId, String groupId)
  - User joins a group. There will be two types of groups, open and closed. The user can freely join open groups, but needs an invitation for closed groups. The group visibility is determined by the group type. In any case, the SLA method is called by the PA only after the user has successfully joined the group at JXTA level.
- Collection<Object> launchQuery(String queryId, Map<String, String> queryParams)
  - Perform the query identified by queryId using the given set of parameters.
  - Returns a matching object.
- void leaveGroup(String userId, String groupId)
  - Called by a user to either leave a group or by the group owner to delete another user from the group.
- String login()
  - Performs login. Logs in the current user. Initializes the connection to the Central and Local matching systems. Loads the CMS server URL and repository ID from the configuration file. Each individual user has a custom local repository. A user must login before using the system. A logout must be performed before a new user can login.

- boolean logout()
  - Performs logout, closes the repository connections and shuts down LMS as well as CMS repositories.
  - Returns true if success false if failure.
- void setNotificationRead(String notifId)
  - Sets the corresponding notification as read.
- void updateActivity(Activity activity)
  - Updates the activity information. Only the creator of the activity can update the activity.
- void updateBlogPost(BlogPost post)
  - Updates the blog post information. Only the creator can update the blog post. It does not matter if it is a user blog or group blog that is updated.
- void updateEvent(Event event)
  - Updates the event information. Only the creator of the event can update the event.
- void updateGroup(Group group)
  - Updates the group information. Only the creator of the group can update the group.
- void updateMedicalProfile(String userId, MedicalProfile medicalProfile)
  - Updates the medical profile information. A medical profile will be created as soon as a user is created and will be deleted when the user is deleted.
- void updateService(Service service)
  - Updates the service information. Only the creator of the service can update the service.
- void updateUser(User user)
  - Updates the information on a user. The passed user must includes a valid Id.
- void acceptFriendRequestFrom(String userId)
  - Accepts a friendship request as notification.
- void deleteFriend(String userId)
  - Deletes a friendship between two users.
- Collection<User> getFriends(String userId, int limit, int offset)

- A list of all friends for a specific user.

- FriendState getFriendState(String user1, String user2)
  - Gather the friendhip status of two users.
- String requestFriend(String userId, String message)
  - Sends a friendship request as notification.
  - Returns the created notificationId.
- void unrequestFriend(String userId)
  - Revokes a friendship request notification.
- void acceptCareconsumerRequestFrom(String targetId, String requesterId)
  - Accepts a careconsumer request as notification.
- void acceptCaregiverRequestFrom(String targetId, String requesterId)
  - Accepts a caregiver request as notification.
- void deleteCaregiver(String user1Id, String user2Id)
  - Deletes a caregiver connection between two users.
- void deleteCareconsumer(String user1Id, String user2Id)
  - Deletes a careconsumer connection between two users.
- Collection<User> getCareconsumers(String userId, int limit, int offset)
  - Returns a list of all careconsumers for a specific user.
- Collection<User> getCaregivers(String userId, int limit, int offset)
  - Returns a list of all caregivers for a specific user.
- CareState getCareState(String user1, String user2)
  - Gather the care related status of two users.
- String requestCaregiver(String requesterId, String targetId, String message)
  - Sends a caregiver request as notification.
  - Returns the created notificationId.

- String requestCareconsumer(String requesterId, String targetId, String message)
  - Sends a careconsumer request as notification.
  - Returns the created notificationId.
- void unrequestCaregiver(String requesterId, String targetId)
  - Revokes a caregiver request notification.
- void unrequestCareconsumer(String requesterId, String targetId)
  - Revokes a careconsumer request notification.
- Long getTopicSearchCount(String userId, String topic)
  - Returns the number of times a user has searched a topic.
- Collection<String> getGroupTopics(String groupId)
  - Return a list of topics for a certain group or all possible topics when groupId has a null value.
- Collection<String> getInterestTopics(String userId)
  - Returns a list of interest topics for a certain user or all possible interest topics when userId has a null value.
- Collection<String> getQueriedTopics(String userId)
  - Returns a list of queried topics for a certain user or all possible queried topics when userId has a null value.
- void raiseTopicSearchCount(String userId, String topic)
  - Raises the counter for a certain searched topic by 1 for a user.
- void bookmarkService(String userId, String serviceId, boolean add)
  - According to the value of 'add', adds or removes a pa:bookmark between user and service.
- Collection<Service> getUserBookmarkedServices(String userId, int limit, int offset)
  - Get all services bookmarked by a user.
- Collection<Service> getUserOfferedServices(String userId, int limit, int offset)
  - Get all services a user offers.
- boolean isBookmarked(String userId, String serviceId)

- Returns whether a user bookmarked a service.

- Collection<User> searchUsers(Map<String, String> queryParams)
  - Search users according to query parameters.
- Collection<Group> searchGroups(Map<String, String> queryParams)
  - Search groups according to query parameters.
- Collection<Service> searchServices(Map<String, String> queryParams)
  - Search services according to query parameters.
- String notifyPendingCommunication(String originator, String receiver, NotificationType notificationType, Long timestamp, String groupId, String message)
  - Notifies about pending communication. If there is already a Notification of type ChatMessage with these originator and receiver, do nothing else, create it. If there is already a Notification of type ChatMessage with these groupId and receiver, do nothing else, create it. If there is already a Notification of type VideoCall with these originator and receiver, do nothing else, create it.
  - Returns id of the new notification, or null if none was created.

3.2 Suggestions logic

In this section we describe our implemented suggestion mechanism. Part of the ontology is dedicated to topics. These topics come in three different flavors. Topics of interest of users, groups and topics a user searched for. These three mechanisms are the base for the suggestion system and obviously improve over the time as the machine learns more about the users and users interests. The goal is to automatically recommend fitting users and groups to other users. Also topics can be recommended as the list of available ones grows with the number of user searches. Therefore the SLA suggests following things:

- List of new topics for a user to search for and to add to his or her profile.
- Fitting groups based upon topics of the group.
- Fitting users, who can be added as friends, based upon their topic of interest profile as well as their search history.
- Fitting activities and events within groups.

These are realized by the getTopicSearchCount(), getGroupTopics(), getInterestTopics(), getQueriedTopics() and Collection<Object> get Suggestions().

4 Installation and configuration

The source code for the SLA component is available from the following source:

- https://subversion.assembla.com/svn/passist/at.sti2.peerassist.sla/

The package can be exported as an OSGI bundle and uses the following internal OSGI bundles:

1. gr.uoa.di.cnl.peerassist.common
2. org.osgi.framework
3. org.osgi.service.cm
4. org.osgi.service.log
5. org.osgi.util.tracker

Also some external libraries are needed in order to run the component:

1. commons-cli-1.2.jar
2. commons-codec-1.4.jar
3. commons-dbcp-1.3.jar
4. commons-httpclient-3.1.jar
5. commons-logging-1.1.1.jar
6. commons-pool-1.5.4.jar
7. httpcore-4.1.3.jar
8. httpcore-nio-4.1.3.jar
9. javassist-3.11.0.GA.jar
10. junit-4.0.jar
11. openrdf-sesame-2.6.5-onejar.jar
12. slf4j-api-1.6.1.jar
13. slf4j-jdk14-1.6.1.jar
14. commons-beanutils-1.8.0.jar
15. commons-collections-3.2.1.jar
16. commons-lang-2.5.jar

17. ezmorph-1.0.6.jar

18. json-lib-2.4-jdk15.jar

No special configuration has to be considered. The compiled bundle can be run within an OSGI container. For example the Apache felix [1] OSGI container was used within the development phase of the project. However when desired the following values can be changed in the conf/system.properies file within the environment.

1. at.sti2.repository.sesameServerUrl
    - Location of the CMS sesame server.
2. at.sti2.repository.repositoryID
    - Id of the CMS repository on the sesame server.
3. at.sti2.repository.lmsRootDirectory
    - The root directory where to store local matching information.

5 Conclusions

This deliverable describes the implementation of a framework enabling persistent storage of data in the discussed scope of the PeerAssist project. So far, the SLA component integrates nicely with the rest of the system components and has proven to run stable. Improvements for the system include the actual storage of local information as well as improving certain aspects of CMS storage performance.

6 References

[1] Apache Felix OSGI container http://felix.apache.org, accessed 25.11.2012