AAL-2009-2-116

Deliverable D3.2
Title: ELDER-SPACES Supporting API interfaces

Deliverable Type:	**PU**
Nature of the Deliverable:	**R**
Date:	**20/03/2013**
Distribution:	**WP3**
Code:	**<ELDER-SPACES_ORIGO_WP3_D3_1>**
Editor:	**ORIGO**
Contributors:	**Byte, SLG, Cybion, FTB, e-Trikala, Semmelweis**

***Deliverable Type:** *PU= Public, RE= Restricted to a group specified by the Consortium, PP= Restricted to other program participants (including the Commission services), CO= Confidential, only for members of the Consortium (including the Commission services)*

*** **Nature of the Deliverable**: *P= Prototype, R= Report, S= Specification, T= Tool, O= Other*

Abstract: The deliverable providesan assessment of the market possibilities for the Elder Spaces platform and services. This deliverable also provides a market survey and business models that guide to a successful commercialization of our project.

BYTE	Project Coordinator	Greece
ORIGO	Partner	Hungary
FTB	Partner	Germany
e-Trikala	Partner	Greece
SEMMELWEIS	Partner	Hungary
SLG	Partner	Greece
CYBION	Partner	Italy

This page has been intentionally left blank.

DOCUMENT REVISION HISTORY

Version	*Date*	*Modifications introduced*	
		Modification reason	*Modified by*
0.1	20/03/2013	First Draft	ORIGO
1	02/04/13	First version	CYBION
2	05/04/13	Second vesion	ORIGO

Table of Contents

LIST OF FIGURES

1. Introduction

1.1 Overview

Elder-Spaces functionalities will work via the API-s of the system. This solution results in the modularity of the platform and the capibility of easy improvements in the future. That's why the most critical part of a well-working Elder-Spaces platform is the fail-safe, reliable and quick API-s.

1.2 Relation with other tasks and WPs

We specified the common API (PAPI) of Elder-Spaces in WP2 according to the requirements we set in WP1. Further developments of WP4 and WP5 will use this API.

2. Developing Elder-Spaces Services and Applications

2.1 Framework

According to the requirements of Elder-Spaces platform that we specified in D2.3 we chose to use Symfony 2.0 framework to implement the PHP front-end of the site.

2.1.1 About Symfony

Symfony is an open source PHP framework has been improved continuously since 2005. Thanks to the serious changes of version 2.0, all the new features of PHP 5.3 (namespace, late static bindings, etc.) are fully supported. This new version has an absolutely modular construction (with bundles), and thanks to dependency injection, each module can be modified or improved easily. That's why the applications created this way will be organized and flexible, that helps adopting further impovements easier.

Version 2.0 brought significant speed-up aginst the previous version and other concurent frameworks as well. Hence this framework is not just flexible, but also very fast.

An other benefit of the system is it's exhaustive documentation and the serious developer base behind, who assist the development in case of any problem and give feedback or fix the bug immediately. Symfony is an open source project, many developers are improving the framework countinously.

Since this system is a framework, it's much less vulnerable than other “higher leveled” open source systems that weaknesses is actually because of their open source code.

2.1.2 Why Symfony?

Symfony gives the advantages to development from which it can be fast and efficient, and it doesn't reduce the possibility of any imagined feature. Symfony supplies all the basic tasks that ensure the standardness and flexibility of the system being implemented in the way that helps the future independence from the developer company of the system.

Symfony assists in the implementation of the basic features which have to be fast, safe and efficient. Such as:

- *quick entity definitions*
- *ORM (Object Relational Mapper) / DAL (Database Abstraction Layer), built in Doctrine*
- *support and require of MVC (Model-View-Controller) based developing*
- *support of standard form handling*
- *component based approach*
- *user handling and authentication interfaces, that helps to implement unique authentication and authorization solutions easily*

2.2 IMPLEMENTING SERVICES AND APPLICATIONS VIA THE COMMON PORTAL API

The goal of this section is to provide detailed information about the functionality of the software after initial of finished reconciliation.

The particularized functionality provided in this document is the base of the development process and is taken as lead documentation.

2.2.1 REQUIREMENTS

2.2.1.1 FRAMEWORK

The software is developed in Symfony 2.1.3 standard edition

2.2.1.2 ADDITIONAL BUNDLES USED

4.2.1.2.1 Communication

- *HWIOAuthBundle : https://github.com/hwi/HWIOAuthBundle*
  - */vendor/hwi*

This bundle is responsible for the OAuth communication to the server so there are many configuration options to take in consideration during the integration:

- */app/config/config.yml hwi_oauth section*
- */app/config/parameters.yml*
  - *elders_api_key*
  - *elders_api_secret_key*
  - *elders.oauth.rest.url*
- */app/config/security.yml*
  - *security:firewalls:social section:*

4.2.1.2.2 json2object, object2json

- *JMSSerializerBundle: http://jmsyst.com/bundles/JMSSerializerBundle*
  - */vendor/jms*

2.2.1.3 TECHNICAL REQUIREMENTS

- *PHP 5.3.8+*
- *memcache*
- *curl*

2.2.1.4 RECOMMENDATION

The content on the FTP servers should be copied into a local repository to work on it further. There are several extra bundles required/downloaded automatically for the in 2.2.1 and 2.2.2 section mentioned bundles and the integration is already done in this copy.

Some of the settings must be changed to go on with the development in the newly copied repository:

- *change working environment from production (prod) to development (dev)*
  - *in /web/index.php the line $kernel = new AppKernel('prod', false); must be changed to $kernel = new AppKernel('dev', false);*
  - *in /web/index_dev.php the line $kernel = new AppKernel('prod', true); must be changed to $kernel = new AppKernel('dev', true);*
- *memcache settings /app/config/parameters_dev.yml to refer your local settings usually on localhost on port 11211. Memcache is used to store session information as of the production server is a shared server.*

The effect after changing to development environment is that configuration files (/app/config) with _dev.yml postfix will be loaded instead of files with _prod.yml postfixes.

2.2.1.5 TRANSLATIONS

The translation files are prepared like in Symfony 2.x. Look for it in the /src/Elders/PortalBundle/Resources/translations/messages.en.xlf file.

If you like to add new languages you have to change the following files:

1. *Create a new file in the translation folder with the pattern messages.COUNTRYCODE.xlf, where country code is the abbreviation of your country in two characters.*
2. *To refer to this you have to add your country to the available languages for the system in /app/config/parameters.yml file in the elderspaces.supported.locales section.*
3. *The flag image must be provided as well and a flag-COUNTRYCODE class must be placed in the css file as well.*

2.2.2 MAPPING OF REST URLS

To leverage the complexity of rest calls to few steps of configuration changes and the making of some new classes we decided to make a descriptor class, a bridge class and a factory pattern around it. All this can be found in the /src/Elders/RestBundle.

2.2.2.1 MAPPING CONFIGURATION

The descriptor itself works from a mapping which are found in the /app/config/config.yml parameters section, currently prepared maps are:

- *elders.event.descriptor.map*
- *elders.activity.descriptor.map*
- *elders.groups.descriptor.map*

As you can see we group this maps for each kind of service (people, activities, groups etc.) together like a traditional REST API call expects. Each map looks like the following:

group_search:

- ***format:*** *json*
- ***pattern:*** *'/clubs/search'*
- ***required:***
  - *keyword: ~*
- ***optional:*** *[]*
- ***resolver_class:*** *Elders\PortalBundle\Entity\OSGroup*

2.2.2.2 DESCRIPTOR FACTORY

Class: Elders\RestBundle\Descriptor\RequestDescriptorFactory

The factory is responsible from validating the descriptor map and creating descriptor instances from it. Each main service (people, activities, groups etc) contains one descriptor map elders.event.descriptor.map, and within this map there are two descriptor configurations called event_get and event_search.

2.2.2.3 DESCRIPTOR MAPS EXPLAINED

From the above example the intended section are as follow:

- ***group_search:*** *is the descriptor map name (similar ones are group_set, group_get etc.)*
- ***format:*** *is the format to what the rest calls communicate with (currently only json is supported)*
- ***pattern:*** *is the rest uri part we match imagine the full url http://api.elderspaces.iwiw.hu/social/papi/rest/clubs/search (elders.oauth.rest.url parameter from /app/config/parameters.yml)*
- ***required:*** *is the section where all the required fields are listed this can be rest url parts or query_string parts as well e.g.: pattern: '/events/¤userId¤/¤eventId¤'. Here you can see with ¤ sign is the separator for res uri parts (this sign is not required anymore for the query_string parts).*
- ***optional:*** *this section list all optional parameters e.g.: paging parameters:*
  - *startIndex: 0*
  - *itemsPerPage: 10*
- ***resolver_class:*** *the class object which the transfer will be converted from json or to json using the JMS serializer bundle*

2.2.3 REQUEST MANAGERS

Classes

- *Elders\RestBundle\Manager\RequestManager ;*
- *Elders\PortalBundle\Manager\AbstractRequestManager\RequestGroupManager*

These managers are responsible for creating a descriptor from a descriptor factory and using a bridge (physical communication) to get the required content using the access token (oauth).

Each manager gets a bridge and factory injected in it's constructor.

2.2.3.1 CONFIGURATION

For each service there is a factory and a manager prepared in /app/config/config.yml services section:

parameters:

elders.request.manager.groups.class: Elders\PortalBundle\Manager\RequestGroupManager

services:

- ***elders.request.descriptor.factory.groups:***
  - *parent: elders.request.descriptor.factory.abstract*
  - *arguments:*
    - *index_0: %elders.groups.descriptor.map%*

- ***elders.request.manager.groups:***
  - *parent: elders.request.manager.abstract*
  - *class: %elders.request.manager.groups.class%*
  - *arguments:*
    - *index_0: '@elders.request.descriptor.factory.groups'*

2.2.4 FORGE EVERYTHING TOGETHER (CONTROLLER PART)

Each request to the server whether to get information search or push information begins it's life cycle in a controller. Lets see an example how a People Friends List request is resolved:

1. */src/Elders/PortalBundle/PeopleController:overviewAction*
2. *This controller gets an instance from the people manager **elders.request.manager.people** and runs a getFriends() method call to get the friends providing a userid. This userid can be any of our friends id or ours (to get our friends use @me).*
3. ***RequestPeopleManager**->getFriends call will create a descriptor for the **people_get** descriptor map which is included within the **elders.people.descriptor.map** which is provided for the factory **elders.request.descriptor.factory.people** which is injected into the **RequestPeopleManagers** constructor. Each method in the RequestManagers can create a different descriptors or the same descriptor with different configuration using the Factory object injected into the manager. I recommend to have a deeper view in*

Symfony's dependency injection description which is heavily based on configuration of services (which will be transferred into usable objects on first request).

4. *After the descriptor is made (step 3.) we call the bridge to transfer our request to the server where descriptor helps us to create the url with all the required and optional parameters set in step 3. The bridge itself makes the request and signs it with the provided access_token.*
5. *The result of the bridge call should return a json object itself and this json object will be transferred into using the JSM serializer bundle into usable object OSPerson (the full list of objects can be seen in /src/Elders/PortalBundle/Entity/OSxxxxx. To which object it will be transferred depends in the resolver_class (4.2.2.1 Mapping configurations) section of a descriptor map entry configuration.*

2.2.5 PREPARING NEW SERVICES

2.2.5.1 EXTENDING USAGES FOR THE SAME SERVICE GROUP

Existing functionality can be reused with mapping that already exists. You just have to provide new public function in the RequestManager which offers than new set of parameters for the map currently used.

2.2.5.2 *EXTENDING SERVICE GROUPS*

Just provide a new descriptor map entry within the maps e.g. (and further see section above 3.4.1):

group_get:

- ***format:*** *json*
- ***pattern:*** *'/clubs/get'*
- ***required:***
  - *name: ~*
- ***optional:*** *[]*
- ***resolver_class:*** *Elders\PortalBundle\Entity\OSGroup*

3. Development of services for Social Search and Concept Based Recommendations

3.1 GOALS OF SEMANTIC MODEL BASED SERVICES

The Social Search and Concept Based Recommendations (SSCBR) services are the interfaces for the SSCBR system. Two types of services have been implemented in order to integrate the Elder-spaces platform with the two main components of the SSCBR system:

- Knowledge acquisition services
- Recommendation services

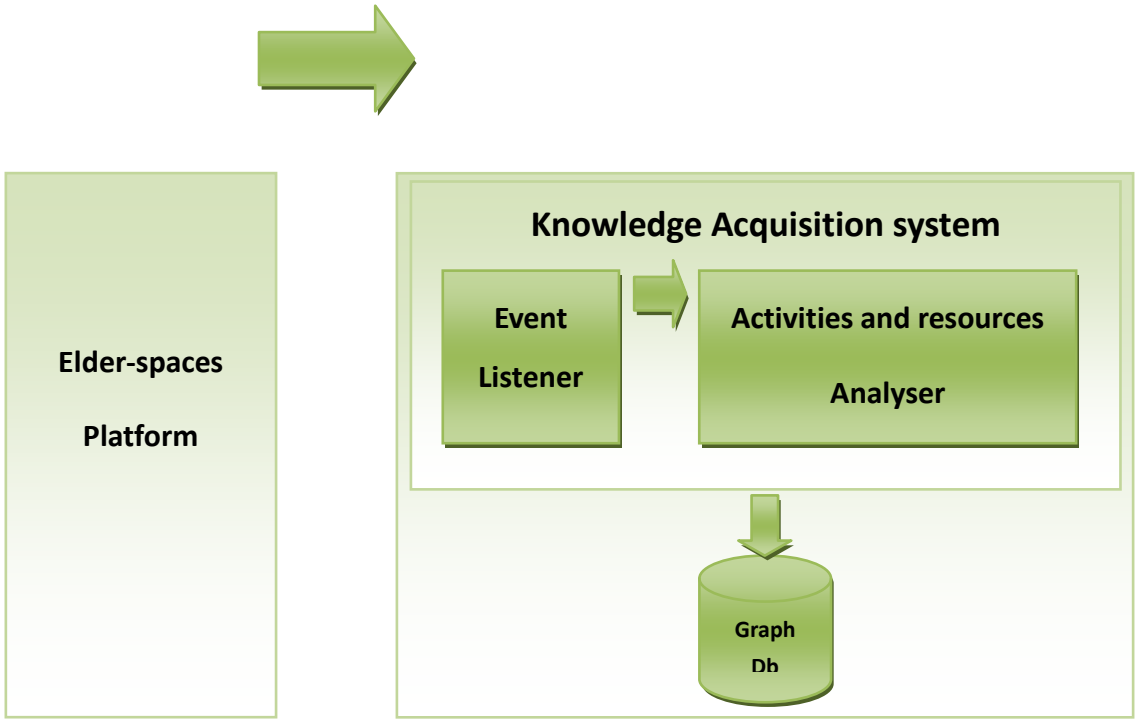
Knowledge acquisition services are responsible to let the Knowledge acquisition system acquire knowledge from the Elder-Spaces platform. The Knowledge acquisition system will be in charge of receiving, translating and enriching data about the resources managed inside the Elder-Spaces social platform, for example; Users, Clubs, Events, Activites, their inter-relations and other activities that involve them.

Recommendation services are responsible to return to the Elder-spaces platform the sets of recommendations for each user: recommendations of new possible friends, recommendations of new events to participate and recommendations of clubs to join where discuss and experiences about topic of interests.

3.2 KNOWLEDGE ACQUISITION SERVICES

The Knowledge acquisition services have been implemented following the specifications of “API Specification appendix - D2.3 Elder-Spaces Platform Architecture”, section 2.10.4 EVENT LISTENER SERVICE. The solution is a dedicated module that can manage specific streams of events; the data flow of the Knowledge Acquisition system is depicted in

Figure 1- Knowledge acquisition architecture



Knowledge acquisition system is integrated with the Elder-spaces platform by a main service: the Event Listener. Knowledge acquisition services comprise also another service that reply about the status of the service.

The Event Listener service has been implemented in order to provide a robust infrastructure that allows the subsystem to be aligned and consistent for what concerns updates of user generated data in the Elder-spaces platform. It works as a messaging system, where the messages producer is the Elder-spaces platform, whenever a user activity is performed and the consumer is the Event Listener service of the Knowledge acquisition system developed by Cybion. Indeed Elder-spaces platform triggers the set of activities that are needed by SSCBR system. We specify thisű as an API extension (Activity Stream for Recommendation Services) in the next (3rd) chapter. The Event Listener service stores the activity in the internal knowledge base of the Social Search and Concept Based Recommendations. In this way the knowledge base is ready to be used by the Knowledge acquisition system.

The first version of the Knowledge acquisition services is up and running at:

http://gaia.cybion.eu:8080/activities/

The **Event Listener** listens activities of Elder-spaces platform through a POST call at:

http://gaia.cybion.eu:8080/activities/rest/activities

where the content should be plain text, containing actually the specified json of the activities.

A working request can be done using the command line from any unix shell using curl

curl http://gaia.cybion.eu:8080/activities/rest/activities *-X POST -d "abc" -H "Accept: application/json" -H "Content-Type: text/plain"*

and the expected JSON output is the following:

{"message":"received activity","status":"OK"}

Another service has been implemented in order to know the status, it is also possible to send GET requests at

http://gaia.cybion.eu:8080/activities/rest/status/now

If the services are up and running receiving a message such as this:

{"message":"services up and running as of '2013-03-03T15:28:14.482Z'","status":"OK"}

3.3 RECOMMENDATION SERVICES

The Cognitive social recommendations are accessible by the Elder-Spaces platform through a layer of services, the Recommendation services that are specified in "API Specification appendix - D2.3 Elder-Spaces Platform Architecture" section 2.10.1-3.

The Recommender system is the module responsible of calculating resources recommendations, keeping them updated and making them available for consumption by the Elder-spaces platform. The recommender sub-system analyzes the knowledge base that is continuously built and enriched by the Knowledge acquisition system in order to calculate the recommendations for each user of the Elder-spaces platform. The recommendations are finally available to be integrated in the Elder-spaces platform through the recommendation services. The data flow of the Recommendation system is depicted in

Figure 2: Recommendation system architecture

The web service layer is up and running at the following URL:

http://gaia.cybion.eu:8080/elderspaces-recommendations-endpoint/

As it has been designed, three main services have been implemented returning recommendations of friends, events and clubs. The services answer to plain HTTP GET requests, received by the Elder-Spaces social platform at:

http://gaia.cybion.eu:8080/elderspaces-recommendations-endpoint/rest/recommendations/friends/{userId}

http://gaia.cybion.eu:8080/elderspaces-recommendations-endpoint/rest/recommendations/events/{userId}

http://gaia.cybion.eu:8080/elderspaces-recommendations-endpoint/rest/recommendations/clubs/{userId}

Here follows an example of answer to a request for user with id 1. It shows the message of the service, encoded in JSON, that returns three recommended users as possible friends:


```
{"object":
  {"entries":[
        {"displayName":"Elfogad,Dezsaz˜",
"thumbnailUrl":"http://thn1.elderspaces.iwiw.hu/0101//user/00/00/00/10/4/user
_104_1238767117477_tn1","id":"104:elderspaces.iwiw.hu"},
        {"displayName":"Maria Orbatal˜",
"thumbnailUrl":"http://thn1.elderspaces.iwiw.hu/0101//user/00/00/00/11/0/user
_110_12387671174323_tn1","id":"110:elderspaces.iwiw.hu"},
        {"displayName":"Marko
Dienigâ€˜","thumbnailUrl":"http://thn1.elderspaces.iwiw.hu/0101//user/00/00/0
0/11/2/user_112_123876711432741_tn1","id":"112:elderspaces.iwiw.hu"}],
  "startIndex":0,
  "totalResults":3},
  "message":"Friends recommendations",
  "status":"OK"}
```

4. API Extensions

4.1 INTRODUCTION

We designed and specified a sophisticated common API (PAPI) for the Elder-Spaces platform is WP2 with all the required entities and services of the basic functionality and the applications as well. With such a wide set of entities and services many new and various features or applications with mixed social functionalities can be implemented into the platform via the API easily.

There was only one need what the common API couldn't support: the intelligent recommendation module which needs to profile the user by it's behavior on the site. So we specified an Activity Stream service that provides a special meta data flow of the user's activities for the recommendation system.

4.2 ACTIVITY STREAM FOR RECOMMENDATION SERVICES

Activity Stream is responsible for the communication between the Elder-Spaces platform and the Social Search and Concept Based Recommendation system. The specification details the serialization of the stream of social activities using the JSON format. The specification is based on the activity stream standard specification1, adapting entities and verbs to the requirements of the Elder-spaces context. In its simplest form, an activity consists of an actor, a verb, an an object, and a target. An activity tells the story of a person performing an action on or with an object. An Activity Stream is a collection one or more individual activities.

4.3 BASIC ENTITIES DESCRIPTION

In this section are decribed each entity that will be involved in the activity stream. We willl reference these entities in the ActivityStream between the Elder-spaces platform and the Social Search and Concept Based Recommendation system.

4.3.1 PERSON

```
{
        "id":"15247988:elderspaces.iwiw.hu",
        "objectType":"person",
        "displayName":"Portál András [Papi]",
         "thumbnailUrl":"http://thn1.elderspaces.iwiw.hu/0101//user/00/00/00/10/4/user_104_
        1238767117477_tn1",
}
```

4.3.2 ACTIVITY

```
{
```

"objectType":"activity",
"body": "Zsír",
"postedTime": 1352728621000,
"bodyId": "CLUB_MSG_FEED_BODY",
"id": "00e8aa7450a1002d1e12b287",
"mediaItems": [],
"title": "Erik ",
"userId": "15247988:elderspaces.iwiw.hu",
"titleId": "CLUB_MSG_FEED"
}

4.3.3 CLUB

{
"objectType":"club",
“id”:”100:elderspaces.iwiw.hu”,
“name”:”Klub neve”,
“description”:”Klub leírása”,
“shortDescription”:”Klub rövid leírása”,
“category”:”HOBBY”
}

4.3.4 EVENT

{
"objectType":"event",
"id": "1181:elderspaces.iwiw.hu",
"shortDescription": "Jemand musste Josef K. verleumdet haben, denn ohne",
"name": "Im Gegensatz zu früheren Webseiten müssen wir zum Beispiel nicht mehr"
}

4.4 TRIGGER CALLS

This section includes the activities performed by users in Elder-spaces that will be triggered to the Social Search and Concept Based Recommendation system. For each activity it has been defined a tuple of four or five elements: verb, actor, object, target and published. Where it was possible, it has been kept the verb definitions from the Activity Base Schema (Draft)1.

4.4.1 SEND FRIEND REQUEST

verb: request-friend, MUST

actor: Person, MUST

object: Person, MUST - the requested friend

published: "2011-02-10T15:04:55Z", MUST

4.4.2 ADD NEW FRIEND

verb: make-friend, MUST

actor: Person, MUST

object: Person, MUST - the new friend

published: "2011-02-10T15:04:55Z", MUST

4.4.3 DELETE FRIEND CONNENCTION

verb: remove-friend, MUST

actor: Person, MUST

object: Person, MUST - the friend to remove

published: "2011-02-10T15:04:55Z", MUST

4.4.4 MODIFY PROFILE DATA

verb: update, MUST

actor: Person, MUST

object: Person, MUST - the modified profile

published: "2011-02-10T15:04:55Z", MUST

4.4.5 DELETE USER

verb: delete, MUST

actor: Person, MUST

object: Person, MUST - the deleted person

published: "2011-02-10T15:04:55Z", MUST

4.4.6 POST ACTIVITY

verb: create, MUST

actor: Person, MUST

object: Activity, MUST - the posted activity

published: "2011-02-10T15:04:55Z", MUST

4.4.7 DELETE ACTIVITY

verb: delete, MUST

actor: Person, MUST

object: Activity, MUST - the deleted activity

published: "2011-02-10T15:04:55Z", MUST

4.4.8 CREATE CLUB

verb: create, MUST

actor: Person, MUST

object: Club, MUST - the created club

published: "2011-02-10T15:04:55Z", MUST

4.4.9 MODIFY CLUB

verb: update, MUST

actor: Person, MUST

object: Club, MUST - the modified club

published: "2011-02-10T15:04:55Z", MUST

4.4.10 DELETE CLUB

verb: delete, MUST

actor: Person, MUST

object: Club, MUST - the deleted club

published: "2011-02-10T15:04:55Z", MUST

4.4.11 JOIN TO CLUB

verb: join, MUST

actor: Person, MUST

object: Club, MUST - the created club

published: "2011-02-10T15:04:55Z", MUST

4.4.12 LEAVE A CLUB

verb: leave, MUST

actor: Person, MUST

object: Club, MUST - the club from where the actor left

published: "2011-02-10T15:04:55Z", MUST

4.4.13 POST ACTIVITY TO A CLUB

verb: create, MUST

actor: Person, MUST

object: Activity, MUST - the posted activity

target: Club - the club where the activity have been created

published: "2011-02-10T15:04:55Z", MUST

4.4.14 DELETE CLUB ACTIVITY

verb: delete, MUST

actor: Person, MUST

object: Activity, MUST - the deleted activity

target: Club - the club where the activity have been deleted

published: "2011-02-10T15:04:55Z", MUST

4.4.15 CREATE EVENT

verb: create, MUST

actor: Person, MUST

object: Event, MUST - the created event

published: "2011-02-10T15:04:55Z", MUST

4.4.16 MODIFY EVENT

verb: update, MUST

actor: Person, MUST

object: Event, MUST - the modified event

published: "2011-02-10T15:04:55Z", MUST

4.4.17 DELETE EVENT

verb: delete, MUST

actor: Person, MUST

object: Event, MUST - the deleted event

published: "2011-02-10T15:04:55Z", MUST

4.4.18 RSVP RESPONSE TO EVENT

verb: rsvp-yes/rsvp-no/rsvp-maybe, MUST - according to the created RSVP status

actor: Person, MUST

object: Event, MUST - the created event

published: "2011-02-10T15:04:55Z", MUST

4.4.19 POST ACTIVITY TO AN EVENT

verb: create, MUST

actor: Person, MUST

object: Activity, MUST - the posted activity

target: Event - the event where the activity have been created

published: "2011-02-10T15:04:55Z", MUST

4.4.20 DELETE EVENT ACTIVITY

verb: delete, MUST

actor: Person, MUST

object: Activity, MUST - the deleted activity

target: Event - the event where the activity have been deleted

published: "2011-02-10T15:04:55Z", MUST