Specific Targeted Research Projects (STReP)

SOCIOTAL

Creating a socially aware citizen-centric Internet of Things

FP7 Contract Number: 609112

WP1 – Socially-aware citizen centric architecture and community APIs

Deliverable report

Contractual date of delivery: 31/08/2014
Actual submission date: 31/07/2014

Deliverable ID:	**D1.3.1**
Deliverable Title:	**First version of API Specification**
Responsible beneficiary:	CRS4
Contributing beneficiaries:	**CRS4, UC, CEA, DNET, UNIS, UMU**

Start Date of the Project: 1 September 2013 Duration: 36 Months

Revision: 1.0
Dissemination Level: Public

Document Information

Document ID: SOCIOTAL_D1.3.1_v1.0.docx
Version: 1
Version Date: 31/07/2014
Authors: Davide Carboni, Alberto Serra, Antonio Pintus (CRS4), Srdjan Krco, Nenad Gligoric (DNET), Jorge Bernal Bernabé, Jose Luis Hernández (UMU), Ignacio Elicegui Maestro, Carmen López, Luis Muñoz Gutiérrez, Luis Sánchez González (UC), Michele Nati, Niklas Palaghias (UNIS), Olivier Savry (CEA)
Security: Public

Approvals

	Name	Organization	Date	Visa
Project Management Team	Klaus Moessner	UNIS		
Internal reviewer	Antonio Skarmeta	UMU	04/07/2014	
Internal reviewer	Dionysia Triantafyllopoulou	UC	10/07/2014	

Document history

Revision	Date	Modification	Authors
0.0.1	19 May 14	First Draft	CRS4
0.1	27 May 14	TOC and initial assignament	CRS4
0.2	03 Jun 14	Contribution for Section 2 and Section 3	CRS4 + DNET + CEA + UMU
0.3	12 Jun 14	General Contribution	CRS4 + DNET + CEA + UMU
0.4	13 Jun 14	Section 1 and VE service communities removed from section 4	CRS4
0.5	18 Jun 14	Section 3 and Section 4	CRS4 + UC
0.6	19 Jun 14	General Contribution	DNET
0.7	19 Jun 14	AP T1.3 call	CEA + UC
0.8	23 Jun 14	Section 1	CRS4
0.9.3	30 Jun 14	All partners contribution almost completed	all
0.10	04 Jul 14	Section1 – Conclusions and CRS4 review	CRS4
0.11	04 Jul 14	Version for Internal review	CRS4
0.24	11 Jul 14	Version reviewed and final contributions added	all
1.0	31 Jul 14	Final version	CRS4

Content

Executive summary

1.1 Description of the deliverable content and purpose

The goal of this task (Task T1.3) is to specify a first overview of APIs (application programming interfaces) that will expose a first set of the required functionalities of SocIoTal system to application developers. This task collects information from the current status of the SocIoTal architecture (task T1.2) and from enablers and components under specification in work packages WP2/3/4. At this stage the API definition does not capture all low level aspects because some of them will be driven by some technological choices that cannot be considered stable yet. As soon as data types, programming languages, and exchange protocols will be more consolidated for each functional component, the API definition will evolve from an overview to a full technical documentation complete in all details. This will happen within M24, at the milestone of the project, where the API will play its role in SocIoTal pilot cases and used for the overall evaluation**.**

The deliverable is structured as follows:

Section 1 – reports a general overview of the SocIoTal API; in particular, it introduces a semi-formal notation to describe them in a uniform way, concerning: naming conventions, parameters and data types, returned values, entities representations, scope, description and access, and API provider as well.

Sections 2 to 3 – provide the API specification for all the envisioned SocIoTal modules. Specifications comply with the notation introduced in Section 1. It is worth to remark that to focus on key challenges and to not reinvent the wheel, the SocIoTal project will reuse where possible existing functionalities implemented by third-party projects and platforms. Such functionalities may range from low-level devices virtualization, physical communication protocols, publish/subscribe mechanisms, and storage services.

In this respect, Section 4 contains the estimated prerequisites from third party platforms that SocIoTal modules expect to find. Also those functions are reported in Section 4 in form of API, but as these are not implemented within SocIoTal, details are purely indicative and may change according to the real API provided by selected third-party components. The work of third-party platform(s) selection is undergoing in task T1.2

1.1.1 Contribution of the Deliverable to the overall project objectives

The main SocIoTal objective is to unleash the full potential of IoT, going beyond the enterprise centric systems and moving towards a citizen inclusive IoT in which IoT devices and contributed information flows provided by people are encouraged. To lower the barrier of IoT bottom-up development and adoption, some key aspects must be tackled: security, control, transparency and simplicity. These aspects are addressed in different blocks of the SocIoTal architecture and are also integrated and made available by means of tools.

An important aspect of any architecture is specification of the interfaces between its components. WP1 specifically addresses the specification of the interface offered to the users, i.e. those developing solutions based on the SocIoTal architecture. API specification allows to define and expose all the SocIoTal architecture functionalities enabling the development of IoT applications on top of them.

The architecture and the APIs will be used to guide the research and development work in WP2, WP3, WP4 and WP5

This deliverable addresses the objective **O1.4: To specify required application programming interfaces (APIs)** related to task *T1.3: Open APIs for service development and IoT device integration (M7- M36)*, that will enable development of services on top of the SocIoTal architecture, having in mind that the targeted users are public at large.

Section 1 - Overview of SocIoTal API

Programming interfaces are a key element in a project with many components that aims at being open to third party applications developers. Thus, the function of programming interfaces is twofold; on one hand it allows to better define the interdependencies among different modules and work packages inside the SocIoTal project itself; on the other it allows to document with a concise set of entry points, the ways an external application could make use of SocIoTal functionalities.

1.1 Introduction

The identification and definition of Programming Interfaces in Task T1.3 is strictly dependent on the task T1.2 running in parallel. The latter is about the definition of the whole SocIoTal architecture in terms of requirements, information flows, functional components, and deployment strategies. In some respect, the present document documents with more details, the interaction between components already identified in task T1.2. Thus, the methodology used to retrieve the API is mainly based on a strict communication and alignment with the activities of software architecture design. Given the fact that the two tasks run in parallel and have the same deadline to produce their first results, the APIs in this document are expressed in form of overview using a mix of formal and informal descriptions. To better classify APIs, some properties are defined such as the scope of access and the type of expected concrete protocol/formalism in which the API and their final implementation will be available. In the following sections, more details about the format, scope and access are described.

1.2 Note on format

The specification so far is still informal because the current deliverable gives an overview of APIs as a stable version of the SocIoTal architecture is not yet available at the time we are writing this deliverable. Nevertheless some minor formalism is given as follows:

ThisIsAFunction (arg1,arg2,…,argN) : Result
Description of “this is a function”

If not specified, type of arguments and return values are generic object references with unspecified attributes. When attributes are known, the object represented in the form {attr1, attr2, ...}. If arguments or return values are list or arrays they are represented as [item,item,....]

ThisFunctionReturnsAList (arg1,arg2,…,argN) : [val1,val2,…]
Description of …

In the example below the function returns an object in form of multiple key,value attributes. Only the name of keys is reported.

ThisFunctionReturnsAnObj (arg1,arg2,…,argN) : {key1,key2,…}
Description of …

If a function does not specify any return value it means it produces a side effect without returning a value.

Create-Read-Update-Delete operations of a resource or entity are often grouped in a single view like below:

addFoo(Foo) **getFoo(fooId):Foo** **listFoo(): [Foo1,Foo2,…]** **updateFoo(Foo)** **deleteFoo(fooId)**
This set of methods aims to handle the management of Foo objects

In this version of the API it is not specified the mechanism for exception raising or error messaging. It is silently assumed that in synchronous remote calls like RESTful services, in case of error a descriptive body is returned and the right HTTP status is used to classify the error.

Example of RESTful body error response
Status: 403 Body: {“error”: “Your request is wrong, please check parameters”}

For local API (like library linking) will be used the common exception rising (which depends on the actual programming language) for synchronous calls, and callback parameters for asynchronous calls.

1.3 Note on description

In this document APIs are grouped by a description, the scope, the access type, and the reference implementation provider. Such information is structured as follows:

Description: the description of the set of APIs
Scope: scope of the API group.

- **Private**: the APIs are used only by one SocIoTal module and not exposed elsewhere
- **Internal**: the APIs are needed to different SocIoTal modules to integrate (eg. from Wp2 to Wp3)
- **Public**: the APIs are exposed to 3-rd party applications.

Access: it is the access of the APIs and describes the type of linking (e.g. web, local library, undefined…)
Reference Implementation Provider: the internal SocIoTal WP where the API is developed (e.g. SocIoTal within WP2/3)

Section 2 - Security, Trust, and Reputation API

2.1 Security Group API

The API exposed in this section refers to the set of functions required to cope with the security and privacy mechanisms adopted in SocIoTal framework. It encompasses methods to deal with different areas of security such as Authentication, Authorization, Identity Management, Trust and Reputation.
The API is expected to implement the novel security and privacy techniques for IoT that are being devised and applied in the scope of the project, and defined in WP2 and WP3. These include privacy preserving identity management, capability and context based access control, context based trust and reputation quantification, secure data sharing within opportunistic bubbles, and so on.

2.1.1 Authorization and authentication API

Description:
Authorization API exposes functions for making authorization decisions based on access control policies. These polices define the permissions that a subject (smart object or user) has over certain target resources (e.g. IoT service).
The authentication API exposes the functions for authenticating users and smart objects based on the provided credentials. The credential can be in form of login/password, shared key, digital certificate.

Scope: Internal
Access: local library
Reference Implementation Provider: SocIoTal within WP2

addAuthzPolicy(authzPolicy) **getAuthzPolicy(authzPolicy_id):AuthzPolicy** **updateAuthzPolicy(authzPolicy)** **deleteAuthzPolicy(authzPolicy_id)**
This set of methods aims to handle the management of Authorization policies, which drive the access control in the framework. Authorization policies are envisaged to be defined in XACML format.
authenticate(credential):assertion
This method allows authenticating users and smart objects based on the provided credentials. The credential can be in form of login/password, shared key, digital certificate. As a result, an assertion is generated to be used afterwards to declare that a specific subject was authenticated successfully by the Issuing authority
verifyAuthNAssertion(authn_assertion):valid/invalid
This method of the Authentication component can be invoked by entities that can verify that an authentication assertion obtained previously is correct.
authorize(subject, resource, action, context): AuthorizationResponse
This function aims to check the permission of a subject to perform an action over a resource, taking into account the current context.
authorize(subject, action, resource) : authzToken
This function is used to generate an authorization token. An authorization request could be made with the optional parameters of *subject*, *resource* and *action* in order to infer the privileges to be embedded in the authorization token

verifyAuthZToken(authz_token):Permit/Deny
This function is used to check the validity of an authorization token. Specifically, it should check at least: ▪ The token is valid and have not expired ▪ The requested action matches a specific privilege in the token ▪ Contextual conditions are fulfilled ▪ Cryptographic verification (e.g. issuer's signature)

2.1.2 Group Manager

Description:
The Group Manager API enables sharing information, in a secure and private way, with those groups of entities (communities or bubbles) which satisfices certain particular set of identity attributes values. These particular sets of attributes are represented by attribute sharing policies which are influenced by context information where the data sharing is being performed.

Scope: Internal
Access: Local library
Reference Implementation Provider: SocIoTal within WP2

AddSharingPolicy (sharingPolicy) **GetSharingPolicy() : sharingPolicy** **UpdateSharingPolicy (sharingPolicy)** **DeleteSharingPolicy ()**
This set of functions aim to manage data sharing policies employed to share information securely within opportunistic bubbles using CP-ABE schema. The function addSharingPolicy(sharing_policy,cpabe_policy) adds a *sharing_policy* to a policy set, specifiying the corresponding *cpabe_policy* to be used in case such *sharing_policy* is successfully evaluated. The sharing policies format is still under designing phase
getSharingKey (attributes) : CPABEKey$_{attributes}$
It returns a *CPABEKey* associated to a specific set of *attributes*. A mechanism to prove the entity possesses such set of attributes is required (e.g. X.509 certificates or anonymous credential systems)
share (sharingPolicy, context) : CPABEPolicy
Based on a previously defined *sharingPolicy* and the current *context* in which sharing transaction is going to be done, it returns a *CPABEPolicy* that will be used by *encryptData* function.
encryptData (data, CPABEPolicy) : ciphertext
It takes the *data* to be encrypted, and a *CPABEPolicy* representing subsets of attributes which are allowed to decrypt *data*. It returns *ciphertext* containing *CPABEPolicy*.
decryptData (ciphertext, CPABEKeyattributes): data
It takes the *ciphertext* to be decrypted, and a *CPABEKey$_{attributes}$* representing a secret key with an associated set of attributes. It returns *data* in the case *CPABEKey$_{attributes}$* satisfies the *CPABEPolicy* which is contained in *ciphertext*

2.1.3 Identity Management

Description:
The Identity Management API exposes the functionalities of Identity Management module and is responsible for managing the identities of the smart objects and users. The module is

able to take into account privacy concerns to manage subjects credentials (from users or smart objects), in a privacy-preserving way relying on anonymous credential systems

Scope: Internal
Access: Local library
Reference Implementation Provider: SocIoTal within WP2

getCredential(Attributes):Credential
This method is invoked by users to obtain a credential from an Issuer. The credential can be an attribute container representing a real o partial identity. The attributes values to be included in the credential are needed. Credentials and the attributes follow a particular structure given by the credential specification.
createCredentialProof(credential, proofSpecification): Proof
This method is use to generate a proof of having credential given a credential. The credential proof structure, which defines the set of attributes and statements the user want to prove, has to be specified. It returns a cryptographic proof of possession of a credential
verifyCredentialProof (credential_proof):valid/invalid
This method is used by the verifier (the service being accessed) to ensure that the user satisfices the required identity attributes. The user credential proof to be validated is passed as parameter. The method returns a Boolean indicating it the proof is valid or not.
createPseudonym(assertion, context):pseudonym
This method is used to generate a pseudonym to a certain subject. Before obtaining a pseudonym, the user should have been previously authenticated, that is why the authentication assertion obtained in the authentication is needed to create the pseudonym.
createGroupPseudonym(assertion, context):pseudonym
This method is used to create a pseudonym for a group of subjects. The pseudonym of the group should be then shared securely within the subjects of the group.
resolvePseudonym(pseudo_ID):identityID
This method is used to resolve the real identity ID given a pseudonym.
LocationBasedPseudoGeneration(LocationData) : LBPseudonym
This function is a part of the “Identity Management” functional component. It is able to generate a pseudonym thanks to the Location data of the user or the device.

2.2 Trust and Reputation API

Description:
The API described in this section provides functions for trust and reputation management for the algorithm developed within SocIoTal WP2**Error! Reference source not found.**. It includes methods for compiling new rules for the algorithm, as well as methods for managing/obtaining virtual entity reputation value by accessing/altering the Trust and reputation rule table.

Scope: Internal
Access: REST Envisioned
Reference Implementation Provider: SocIoTal within WP2

GetTrustAndReputationRule(VE_id): {rules} **AddTrustAndReputationRule(new_register_data)** **RemoveTrustAndReputationRule(VE_id)**
This set of methods is used to manage rules for the Trust and reputation algorithm that is used to compute the Reputation score. Rules can be added as a new key-value pair; removed or looked up by its ID.

GetReputation(VE_id): reputationScore **SetReputation(VE_id, VE_parameter, VE_value)**
This set of methods is used to manage the Reputation score value for the Virtual Entity. setReputation() is a verbose method as it may require additional information for reputation score change depending on the final algorithm used from reputation computation (algorithm may use entity history data which may override Reputation each time a score is generated automatically and not manually by setting the value using this method);as well as VE type which value is to be set. These additional VE parameters are represented as CSV and can be an empty string in case no parameters are needed.

Section 3 - Sociotal Enablers and tools

3.1 Context Manager

Description:
According to the initial architecture of the Context Manager, described in D2.1, the APIs described in this section provide the functions for a Context Provider to Create, Read, Update and Delete in the Context Communication module any given context suitable for the SocIoTal environment. A dedicated Context Inference module, provided by the corresponding Context Provider, will perform the computation of such context.

Scope: Internal
Access: REST envisioned
Reference Implementation Provider: SocIoTal within WP3

In the following CRUD APIs for the Context Provider are defined.

createContext(ContextModel) : ContextID
This method allows a Context Provider to create a specific context into the Context Communication module, represented by the given ContextModel. The method returns a ContextID that allows the Context Provider to identify the given context.
readContext(ContextID) : ContextDescription
This method allows any Context Manager module to retrieve a given context, using the dedicated ContextID.
updateContext(ConextID, ContextDescription)
This method allows the Context Provider to update a given ContextID with the actual context value, when the specified context is identified.
deleteContext(ContextID)
This method allows to remove a given context identified by ContextID.

In the following some basic APIs for the Consumer to Manager interaction are provided.

queryContext(ContextModel) : ContextSpecification
This method allows any Context Consumer to query the Context Manager (and in particular the Context Communication module) for a specific context represented by the provided ContextModel. If a matching context is identified, the corresponding ContextSpecification is provided to the Context Source.
subscribeContext(ContextModel) : SubscriptionID
This method allows any Context Consumer to subscribe to the Context Manager (and in particular the Context Communication module) for a specific context represented by the provided ContextModel. The method returns whether or not the subscription was successful.

The following call is from Context Manager to Context Consumer and assumes that Consumers may expose a RESTful endpoint able to receive notification from the Context Manager

notifyContext(ContextModel) : ContextSpecification
This method is used by the Context Manager (in particular its Context Communication module) to notify all the subscribers for a given ContextModel when the required context is matched. Details about the matching context are provided by the ContextSpecification.

The following APIs describe few methods to invocate the inference of specific contexts by a dedicated Context Inference module provided by SocIoTal platform.

MobilityLearningAndReliability(LocationData) : MobilityPattern
This function is a computation engine fed by **time-stamped absolute location information inputs regarding mobile devices/users**, which can be acquired by any continuous (e.g. GPS/GNSS, Real Time Location Systems, location-enabled Wireless Sensor Networks, WiFi fingerprinting…). As a first outcome, the function can issue mobility habits in the form of the **most probable paths/directions/next-step hop followed by the devices/users**. It can also produce additional quality indicators reflecting the reliability of the mobility learning process it-self, that is to say, the **mobility predictability** (i.e. how much the past estimated trajectories have been spatially dispersed so far, and thus, reflecting if the inferred average mobility behavior is sufficiently representative or not).
LocationBasedPseudoGeneration(LocationData) : LBPseudonym
This function is a part of the "Identity Management" functional component. It is able to generate a pseudonym thanks to the Location data of the user or the device.

3.2 VE Service: Communities (community creation)

Description:
Allows a user (the owner) to create a community to share information and resources with other interested and identified users, according to a defined set of policies. It will provide the methods to add, remove and modify users and resources, as well as to modify the different policies of the community, to adapt them to members' requirements. The API described in this section includes the basic envisioned functionalities and methods for this component.

Scope: Public
Access: REST Envisioned
Reference Implementation Provider: SocIoTal within WP3

CreateCommunity(communityData, owner, policies):community_ID
It defines and creates a community within the SocIoTal platform and returns the Community_ID (later used to reference the community). It will use also internal methods to: - AssignOwner (owner) → define the owner of the community. The owner parameter corresponds to a user/identity object/entity. - AssignPolicies (policies) → define the general policies to share resources within the created community - AssignStorage (size):storage_pointer → from the storage module of the platform, it will create a suitable room (size) to store the corresponding community info and get the pointer to this space. Size parameter can be included in the communityData structure or be internally fixed. The general information of the new created community is given within the communityData structure, according to the community model defined in SocIoTal, and should include data related to the objective of the community, location of their resources, users, etc.
DeleteCommunity(community_ID)
It removes the given community (community_ID) and frees up all their resources.
AddUserRequest(community_ID, user)
It sends a request to the community owner (and/or authorized profiles) to include the requester identity/user (user) within the community (community_ID).
Add/RemoveUser(community_ID, user, profile)
Adds/removes the specified identity (user) to/from the community (community_ID) with the pointed profile (profile). Only the allowed users of the community can add/remove new identities/users to the community, according to the community's policies. To remove a

user/identity, the profile is not needed.
Add/Remove/ModifyResource(community_ID, resource_ID, policy)
Adds/removes/modifies the specified resource/entity (resource_ID) to/from/of the community (community_ID), with the shared policies pointed (policy).
ModifyComunityPolicies(community_ID, new_policies)
Modifies the security default policies (new_policies) of the community (community_ID). These policies define how the resources belonging to the community are going to be shared within their members by default. If any resource need specific policies, these should be defined through AddResource() method.
ModifyCommunityOwner(community_ID, new_owner)
Changes the owner of the community (new_owner). The new-owner parameter corresponds to a user/identity object/entity. This method can be only executed by the currently owner of the community (community_ID)
ModifyCommunityData(community_ID, communityData)
It is used to change any parameter of the community general description, as well as for management purposes (such relocation of community's components, storages, etc.). This method should be only executed by the community owner and/or the allowed user profiles (according to the envisioned modifications).
CommunityInfo(community_ID):communityData
Resolves the community_ID inserted as parameter and returns all information of the community (owner, users, resources, size, etc.) in a structured format (communityData).

3.3 IoT Service: Registering Device Request

Description:
This module enables a user to register a sensor, device or other information source within the SocIoTal platform, creating a new resource, associated to the registering user, and according to a standard resource model (provided by SocIoTal and implemented on the platform). This resource model will describe the registered device, including their capabilities (related to the kind of data it provides) what will assist on Virtual Entities definition and their corresponding associations.

Scope: Public
Access: REST Envisioned
Reference Implementation Provider: SocIoTal within WP3

RegisterResource(owner, resourceData) : {resource_ID}
Collects, directly from the device and/or through a defined user interface, the appropriate (and sufficient) data to fill the provided resource model (resourceData). Once the information is checked, it creates and pushes the command to the platform gateway, addressed to the Resource Directory to perform the registration. The VE Historic Storage and the VE & IoT Service Monitoring will be notified of the new resource by the RD. This method also associates this new resource to its owner (owner), identified by a user object/entity. It returns the resource_ID to be later used in read/write operations, modifications, addition to communities etc.
ModifyResource(resource_ID, owner, resourceData) : {resource_ID}
Modifies (initially) any data, attribute or capability on the resource model describing the pointed resource (resource_ID). It sends the new composed command including the modified resourceData struct to the platform's Resource Directory (where the resource was previously registered) in order to perform the changes and notify the VE Historic Storage and the VE & IoT Service Monitoring. It can be used, for example, to modify the location of a given device once it has been moved, in order to associate its measurements (or observations) to its coordinates. Depending on associations with Virtual Entities, some attributes might not be

suitable to be changed directly through this method. Only allowed users/identities will be able to modify resources and/or some of their attributes (according to defined policies). If the modification process is executed properly, the resource_ID is returned.
RemoveResource(resource_ID, owner): {resource_ID}
Removes (initially) the whole pointed resource (resource_ID). It composes and sends the remove command to platform's Resource Directory (where the resource was previously registered), according to its specifications. The VE Historic Storage and the VE & IoT Service Monitoring will be notified by the RD. Only allowed users/identities will be able to remove resources (according to defined policies). If the removing process is executed properly, the old resource_ID is returned.

3.4 IoT Service: Discover Request

Description:
This discovering service is intended to provide information in two different ways, complementing the Face 2 Face enabler and the location based discovering process:

- From a given "entity" ID (user_ID, community_ID, resource_ID, etc.) it will return the list of available services and/or attributes provided by the identified entity and according to the policies associated to the requester user/identity
- From a given "attribute" (e.g. temperature, "be discoverable", photo sharing, location etc.) it will return a list of "IDs" (user_ID, community_ID, resource_ID, etc.) providing such attribute and available for the requester user/identity according the defined policies.

Scope: Public
Access: REST Envisioned
Reference Implementation Provider: SocIoTal within WP3

DiscoverDevice/VE/User/Community(device_ID/VE_ID/user_ID/community_ID): [resources/attributes/services, …]
From a given entity ID, it triggers the discovering of services and/or attributes the identified entity provides to the requester identity, according to the defined security policies: - DiscoverDevice/Resource/VE (device_ID/resource_ID/VE_ID) → returns a list of available capabilities/services/associations the identified device/resource provides to the identified requester - DiscoverUser (user_ID/identity) → returns a list of available services and resources the pointed user/identity provides to the identified requester - DiscoverCommunity (community_ID) → returns a list of available services and resources within the identified community for the identified requester
DiscoverDevice/VE/User/Community/Service(attribute/service_description): [device_ID/user_ID/community_ID/service_ID, …]
From a given attribute (or set of attributes) or a service description, it triggers the discovering of "entities" providing such services and/or attributes, according to the defined security policies and the identity of the requester

3.5 IoT Service: Subscription Request

Description:
This component provides a connection between the user environment application (or another functional component) and the IoT Broker for the subscribe/notify pattern mechanism. The IoT Service: Subscription Request Component is able to forward notifications from the broker when new events are generated.

Scope: Internal
Access: REST Envisioned
Reference Implementation Provider: SocIoTal within WP4

SubscriptionRequest(requester_ID,[VE_ID/association_ID/community_ID…]): Subscription_ID
It allows an application useror a SocIoTal component (identified by a requester_ID) to request a subscription for any entity/service/observation/event to the IoT Broker. Inside this component it is possible to manage IoT Broker CRUD subscription operation (described in 4.9). A notification is automatically sent to the requester.
SendNotification(subscription_ID, notification_Data)
It will send notification_Data to the listening service, obtained from the subscription_ID

3.6 Metadata Processor

Description:
Focused so far on observations and events provided by the devices registered in the SocIoTal platform, it conforms the raw data gathered, according to the data model defined within SocIoTal platform and pushes the so constructed information to the device gateway.

Scope: internal
Access: REST Envisioned
Reference Implementation Provider: SocIoTal within WP3

CreateObservation(resource_ID, raw_Data) : observationData
Receives raw information (raw_Data) from an identified resource (resource_ID) (e.g. temperature, wind speed, etc.) and according to the data model defined in SocIoTal, builds the observationData structure including the gathered info. It will return structured and formatted data (such a JSON/XML /SensorML file), according to platform requirements.
PushObservation(resource_ID, observationData)
Pushes the formatted observation (observationData), according to platform requirements (in JSON, XML, SensorML, etc.) to the corresponding device gateway. It also includes the ID of the resource (resource_ID) generating the observation, in order to be associated.

3.7 VE Service: Face to Face Interaction Detection

Description:
This VE Service considers information on nearby users' presence and facing direction. Each device estimates the interpersonal distance with its nearby devices, through a novel machine-learning based technique to classify if a user is in proximity to perform face-to-face interaction. Further, the relative orientation of the users is computed through a threshold-

based approach. The knowledge of the users' interpersonal distance estimation and relative orientation computation is combined to infer if the users are performing a face-to-face interaction, allowing the assessment of the users' trust and reputation as well as the estimation of the social relations among devices.

Scope: Public
Access: Local library/Android /REST
Reference Implementation Provider: SocIoTal within WP3

createF2FList(DeviceAddr)
This method creates a list of devices that are facing the device identified by DeviceAddr in the SocIoTal servers. Information is fetched from the DeviceAddr device by querying it at the time of request. The method is asynchronous and will return the list once a F2F discovery has been performed.
readF2FList(DeviceAddr): [Device1, Device2, …]
This method returns the list of devices that have shown a F2F interaction with the considered DeviceAddr at the time of a F2F discovery was triggered. The Device element includes also additional information about the type of faced device, such as SocIoTal identifier and duration of the F2F interaction.
updateF2FList(DeviceAddr)
This method updates the list of devices that have shown a F2F interaction with the considered DeviceAddr, by triggering a new F2F discovery.
deleteF2FList(DeviceAddr)
This method deletes the list of devices that have shown a F2F interaction with the considered DeviceAddr device from SocIoTal servers.

3.7.1 IoT Service: DirectionData

Description:
This IoT Service is used in order to store information regarding the users' facing direction. This will then be used by the VE Service Face to Face Interaction detection in order to infer face-to-face interactions among users. The front direction of participants' torsos as detect by user mobile smartphones worn in different body position is considered as user direction.

Scope: Private
Access: Local library/Android/REST
Reference Implementation Provider: SocIoTal within WP3

readUserFacingDirection(): Direction
This method is internally called by the F2F algorithm running on mobile phones in order to extract user facing direction from mobile phone sensors.

3.7.2 IoT Service: NearbyDevicesData

Description:
This IoT Service is used in order to store information regarding presence of nearby users. This will then be used by the VE Service Face to Face Interaction detection in order to infer face-to-face interactions among users. To detect nearby devices there is a need of obtaining either the absolute or the relative position of the devices in vicinity. Absolute position requires continuous monitoring of the user with positioning sensors such as GPS but are very energy consuming and do not operate indoors. Other absolute position systems exist but require additional hardware. This service uses a new machine-to-machine algorithm to infer device distance using Bluetooth signal strength.

Scope: Private
Access: Local library/Android /REST
Reference Implementation Provider: SocIoTal within WP3

readNearbyDevices(): [Device1, Device2, …]
This method is internally called by the F2F algorithm running on mobile and returns a list of devices with which a possible face-to-face interaction could exist. Only devices discovered and estimated to be within a given distance and with relative orientation suitable for face-to-face interaction, are returned.

3.8 Happiness Index Computation Tool

Description:
Happiness index computation tool provides methods to compute citizens' happiness index based on a set of predefined inputs. Each input is weighted in scale [0,100] and its value represents the impact that the entity has on the final index value. Final happiness index value is given as a mean utility of all provided inputs.

Scope: Internal
Access: REST
Reference Implementation Provider: SocIoTal within WP1

getHappinessIndexInput(VE_id): [input_1, input_2, input_n] **setHappinessIndexInput(VE_id, VE_value, VE_weight)**
This set of methods is used to manage the input(s) of the algorithm used to compute the final happiness index value. VE weight is an integer value that represents an impact that each input have on a finally computed mood happiness index.
getHappinessValue(VE_id): **setHappinessValue(VE_id)**
These methods are used to manage a current value of the virtual entity ID. Virtual entity in this case can be a municipality, city or an area for which a happiness value is computed.

3.9 Visualization module

Description:
The Visualization Tools is an enabler that can visualize a number of different entities by using the Visualization dashboard. All fields of the dashboard are independent and can be controlled individually by visualizing different letters, numbers or even pictures.

Scope: Internal
Access: REST
Reference Implementation Provider: SocIoTal within WP1

setVisualization(dashboardId, VEType, VEValue) **getVisualization(dashboardId): VEValue, VeType**
This set of methods is used to manage the visualization of the dashboard. getVisualization returns the current entity visualize. Visualization of the dashboard can be set by using dashboards ID and VEType and VEValue as parameters. VEType is a types of entity that should be visualized as this information is required from the dashboards in order to visualize different media such as images, text, etc.

3.10 User and Developer Environment API

3.10.1 Process Modelling: Composition (Trigger/Action) API

Description:
These APIs allow a user to create compositions between triggers and actions of different Virtual Entities. These Virtual Entities (VE) are known as **API Plug** in WP4 [2].
An **API Plug** handles the communication abstraction to a **Device** or to a **Service.** In this respect a **Plug** can be seen as univocally related to a **Virtual Entity** in IoT-A terminology.
A **trigger**, registered in a business logic composition, is a condition to be evaluated on the incoming data on the source Virtual Entity. An **action** is the set of operations to be executed on the destination Virtual Entity when the trigger is valid.

Scope: Public
Access: REST
Reference Implementation Provider: SocIoTal within WP4

addComposition(ve_id, data) **updateComposition(composition_id, data)** **deleteComposition(composition_id)** **getCompositionList(ve_id): [composition]** **getComposition(composition_id): composition**
First three methods aim to handle the management of **Composition** objects. A Composition object is composed by a trigger for the source VE and an action for a destination VE. **Data** is a document that contains the information about the creation of a composition, in particular: ve_id of the destination (where the action will be executed) and the trigger logic to be executed. The method **getCompositionList** returns all the compositions for a specific virtual entity and **getComposition(composition_id): composition** returns information about a single composition.

3.10.2 Process Execution: Scheduler API

Description:
These APIs allow to handle the **scheduler**. A scheduler is a process that evaluates and executes VEs business logic compositions on incoming data. This process execution generates a new event and sends triggered data to the connected VE.

Scope: Public
Access: REST
Reference Implementation Provider: SocIoTal within WP4

startScheduler(ve_id)
Start the scheduler process for a Virtual Entity.
stopScheduler(ve_id)
Stop the scheduler process for a Virtual Entity.
getSchedulerStatus (ve_id): {<status_info>}
Return the scheduler status information for a single Virtual Entity

Section 4 - Prerequisites functions from third party platform

API required to be found in selected platform not implemented within SocIoTal tools/platform. The selected platform should provide:

- **Virtual Entity Resolution:** provide functionalities to retrieve associations between VE's and IoT Services.
- **Resource Directory:** will keep track of all registered resources
- **Storage:** a module where identified observations, events and other relevant information to share among users could be kept in a centralized sharing point
- **IoT Broker:** to perform the automatic notifications pushing to those registered applications. Manage publish/subscribe operations.
- **Device gateway:** virtualizes real devices and adapts their low level protocols to HTTP stack.

4.1 VE & IoT Service monitoring

Description:
This component [5] is in charge of automatically find new Associations, which are then inserted into the *VE Resolution* functional component. The new Associations can be derived based on existing Associations, service descriptions (obtained from the *IoT Service Resolution*) and information about VEs.

Scope: Internal
Access: REST
Reference Implementation Provider: Implementations should be provided by chosen platform otherwise by SocIoTal.

assertAssociation(Association):AssociationID
A new association has been found and it is inserted into the VE Resolution functional component.
associationNoLongerValid(AssociationID)
During the monitoring the associations are checked whether the conditions that lead to the creation of the association still hold. If not, this operation is called with the AssociationID as parameter. As a result, the Association will be deleted from the VE Resolution functional block.
associationUpdate(Association)
During the monitoring an association can change some aspects, in this case this operation would update the association.

4.2 Virtual Entity Resolution

Description:
According to IoT-A [3], the VE Resolution will provide the functionalities to the IoT user to retrieve associations between VE's and IoT Services. An association [4] can be summarised as a link between a VE attribute (such as temperature, humidity, speed, etc.) and the IoT service that provides that attribute information (reads and retrieves the temperature, humidity, speed…) or activates it (in case of an actuator). The functionalities of the VE Resolution include the discovery of new and mostly dynamic associations between VE and associated services. If no association exists, the association can be created. The user can also subscribe or unsubscribe to continuous notifications about association discovery that fit

a provided specification of the VE or of the Service. Similar, the User can subscribe or unsubscribe to notifications about association lookup. The VE Resolution will also allow to lookup VE-related services, i.e. search for services exposing resources related to a VE. Finally, it will enable the association management: insert, delete and update associations between a VE and the IoT Services that are associated to the VE.

Scope: Internal
Access: SOAP/REST.
Reference Implementation Provider: IoT-A provides specifications [3]. Implementations should be provided by chosen platform otherwise implemented by SocIoTal.

Discover (VE_type, association_type):[list of VE_IDs, list of associations_IDs]
Discovers new (mostly dynamic) associations between VE and associated services. For the discovery, qualifiers such as location, proximity, and other context information can be considered. If no association exists, it is created.
LookForVEServices (VE_ID, service/attribute):[service_id1,…,]
Searches for services exposing resources/attributes related to a given VE (VE_ID). Type of services/attributes searched could also be pointed (service/attribute type). It returns a list of available IoT Services (services_ids).
insert_association (VE_ID, association_data) **update_association (VE_ID, association_data)** **delete_association (VE_ID, association_data)**
Inserts a new association/Deletes/Updates an existing association between a VE and the IoT Services that are associated to this entity.
(Un)Subscribe (VE_ID, service_type): subscription_ID
(Un)Subscribes the user for notifications about associations/services based on a given VE (VE_ID) and the VEServiceSpecification (within service_type), to be sent to the provided notificationCallback function. A unique SubscriptionID is returned to the subscribing User that can be used to match notifications to the subscription and to unsubscribe.
(Un)Subscribe (association_type): subscription_ID
(Un)Subscribes the user for continuous notifications about associations that fit provided VESpecification and the VEServiceSpecification (within association_type), to be sent to a provided notificationCallback function. A unique subscription_ID is returned to the subscribing user that can be used to match notifications to the subscription and to unsubscribe.

4.3 Virtual Entity Services (Attributes)

Description:
This set of services deal directly with the attributes of the entities (temperature, speed, location, etc.) providing an access to the selected attribute of the pointed VE for requester users/identities. These VE services complement the information provided by the associated (to the VE attribute) IoT Service with information related to the selected VE (entityId, entityType or other attributes belonging to it).

Scope: Internal/Public (depends on the provider platform)
Access: REST envisioned (depends on the provider platform).
Reference Implementation Provider: IoT-A provides specifications [3]. Implementations should be provided by chosen platform otherwise by SocIoTal.

RetrieveData(VE_ID, attribute_ID): structured_Data **DeleteData(VE_ID, attribute_ID)** **StorageData(VE_ID, attribute_ID, structured_Data)**
CRUD for attribute items. VE_ID identifies the Virtual Identity requested and the attribute_ID identifies the selected attribute to read/delete/write. Structure_Data represents the object/structure that handles the attribute information, enriched with entity data. It is used to retrieve the info gathered from the VE as well as to pass attribute info to the platform.

4.4 Resource Directory

Description:
The Resource Directory will keep track of all registered resources belonging to SocIoTal platform, supporting all the resource registration process. It is typically articulated around a database storing information about resources belonging to the system, including their ownership, capabilities, rules, and rights. It will also retrieve information related to resources when required from other component or user. It can be seen as part of the IoT-A Members Functional Component [3].

Scope: Internal
Access: REST.
Reference Implementation Provider: Implementations should be provided by chosen platform otherwise by SocIoTal.

CreateResource (resource_Data):resource_ID
Registers a new resource in the platform. It gets a structured and formatted (according to the resource model defined) resource_Data, registers it and returns the resource_ID to be used when later a reference to this resource is needed. It also notifies the corresponding components (e.g. VE Resolution and VE & IoT Service monitoring) about this resource and it capabilities/attributes, in order to create/update VEs.
FindResource (device_ID/description/capabilities):resource_ID
From a given parameter (device_ID, description, location, capabilities…) it returns a list of resource_IDs mapping that/those parameters
RetrieveResource (resource_ID): resource_Data
It returns the currently stored information (resource_Data) related to the given resource_ID, according to the resource model defined.
Delete(resource_ID)
It removes the given resource (resource_ID). It notifies the corresponding components (e.g. VE Resolution and VE & IoT Service monitoring) about this action, in order to update the VEs' attributes.
Update (resource_ID, resource_Data)
Updates the identified resource (resource_ID) with new data (resource_Data). It also notifies the corresponding components (e.g. VE Resolution and VE & IoT Service monitoring) about this resource and it capabilities/attributes, in order to create/update VEs.

4.5 IoT Service: Resource History Storage

Description:
The platform hosting SocIoTal should provide a storage module where identified observations, events and other relevant information to share among users could be kept in a centralized sharing point (although not all data suitable to be shared will be stored in the platform). This storage module will implement methods to insert (post), delete, update (put), and request (get) stored data.

Scope: Internal
Access: REST
Reference Implementation Provider: Implementations should be provided by chosen platform otherwise by SocIoTal.

SetInfo (source_ID, info)
(Post) method to add new info to the centralized storage (creates a new data record). Source_ID param will identify the source (a resource, a VE, a community, a user …) to which the info will be associated. The info parameter will be composed by structured and formatted data, according to the corresponding data model set by SocIoTal and the platform, e.g. an observation, an event, etc.
GetInfo (info_type, arg1,arg2,...): (Info)
(Get) method to retrieve info_type (observations, events, historical…) according to one or several arguments (source_ID, date, type of measurement/observation, location, date…). It will return selected (and available) info formatted according the corresponding data model.
updateInfo(source_ID, record_info, info)
(Put) method to update an existing record in the centralized storage. Source_ID param identifies the source (a resource, a VE, a community, a user …) to which the original record was associated (to perform a search of the record/s, together with record_info, to be updated). The info parameter will be composed by structured and formatted data, according to the corresponding data model set by SocIoTal and the platform, and will contain the data to perform the update.
deleteInfo (**source_ID, record_info)**
(Delete) method to perform the purging of the existing records that match with the conditions pointed by source_ID and record_info.

4.6 IoT Service: Resolution

Description:
This component [5] provides all the functionalities needed in order to find and be able to contact IoT Services. Also, the IoT Service Resolution gives to Services the capability to manage their service descriptions, so they can be looked up and discovered. The main functionalities of this block are the Resolution, Lookup and Discovery. The Resolution function resolves the Service IDs and returns Service URLs; the Lookup functionality enables to access the service description from the Service ID; and the Discovery functionality finds the IoT Service by providing a service specification as part of the query.

Scope: Internal
Access: REST
Reference Implementation Provider: Implementations should be provided by chosen platform otherwise by SocIoTal.

resolveService(ServiceID):ServiceURL
This operation enables the user of the system to have the address information which enables him to contact the service of interest. This function is called with the ServiceID as parameter, and as a result the Service URL is returned.
lookupService(ServiceID):ServiceDescription
This operation returns the ServiceDescription with the ServiceID as parameter. The ServiceDescription offers information such as whether the service is available in that area, availability in defined time ranges, geographical information, initial conditions of resources, or also the Service URL.

discoverService(ServiceSpecification): ServiceDescription[]
Allows the requester to make a query based on specification of a needed service and without any prior ID knowledge. The related results are returned as an array of ServiceDescription.
insertService(ServiceDescription):ServiceID
Add a new service description to the IoT Service Resolution and it will return the ServiceID.
updateService(ServiceDescription)
Whenever changes occur in the ServiceDescription, the information in the registry must be updated by the IoT Service to allow the continuous service availability.
deleteService(ServiceID)
This operation deletes the ServiceDescription of the Service with the ServiceID as input parameter.

4.7 IoT Service: Post Request

Description:
This component provides all the functionalities needed in order to post data to the platform. IoT Service resolution [4.6] should be used to retrieve the list of available service with description and the service URL. The data can be then posted in the predefined format to the URL retrieved from the IoT Service Resolution.

Scope: Public
Access: REST
Reference Implementation Provider: Implementations should be provided by SocIoTal.

sendData(VE, service_url): VE
This method is used to send the virtual entity to the corresponding Service URL, which will return corresponding HTTP response code

4.8 IoT Broker

Description:
The IoT Broker is a web service endpoint that enables communication between the client, i.e. a data consumer and the server that also provides subscription and data forwarding capabilities.

Scope: Internal
Access: REST.
Reference Implementation Provider: Implementations should be provided by chosen platform otherwise by SocIoTal.

SubscriptionRequest (requester_ID, [VE_ID/association_ID/community_ID…])
It allows an application/user (requester_ID) to request a subscription to any entity/service/observation/event supported by the platform: a user/app can request a subscription to a "VE_Wheather_Station_01" or an observation (Temperature) so, when any change on the "VE_Wheather_Station_01" (a new association is created with a new service or a new measurement is available) a notification is automatically sent to a listening service (specified in CreateSubscription method). In the case of an observation subscription, when any new temperature is reported (by any allowed entity) the requester will also be notified.
CreateSubscription(requester_ID,[VE_ID/association_ID/community_ID/ observation_type…],listening_service_ID): [subscription_ID] **DeleteSubscription(subscription_ID)** **UpdateSubscription(subscription_ID, subscription_Data)**

CRUD for Subscriptions: requester_ID identifies the user/device/app… requesting a subscription; VE_ID/association_ID/community_ID/observation_type… identifies the subject to subscribe to; listening_service_ID provides the URI of the service to be notified; subscription_ID identifies the created subscription, to be used when a notification happens and to call delete/update procedures.
SendNotification(subscription_ID, notification_Data)
It will send notification_Data to the listening service, obtained from the subscription_ID
SetData(VE_id, VE_val) **GetData(VE_id): VE_val** **DataForward(VE_id, VE_type, VE_value)**
This set of methods is used to manage value of the virtual entity by its ID. DataForward method enables VE forwarding between platform components as well as forwarding to the third-party external components.

4.9 Device Gateway

Description:
Connected to the SocIoTal metadata processor provides mechanisms to virtualize real devices and adapts their low level protocols to high level stack (e.g. HTTP).

Scope: Internal
Access: REST.
Reference Implementation Provider: Implementations should be provided by chosen platform otherwise by SocIoTal.

GetDevices() : [deviceId,deviceId,…] **Register/Read/Update/Remove Device (deviceId): DeviceInfo** **GetEvents(deviceId): [event,event,…]** **SubscribeDeviceEvents(deviceId, listener)** **PostData(DataItem, deviceId)** **GetData(deviceId):[DataItem,DataItem, …]**
This set of methods is used to manage physical devices connected to a Gateway that virtualizes and abstracts low level details up to application level. In this group of functionalities you may find API calls to register and update devices actually connected, and API calls to send/read data item to devices and to get events from devices. It is expected that the Gateway is able to bring such functionalities at HTTP level, but also other communication protocols like MQTT or WebSocket may apply.

Section 5 - Conclusion

This document reports the Application Programming Interface (API) specification of the SocIoTal platform and it addresses the objective **O1.4: To specify required application programming interfaces (API)** related to task *T1.3: Open APIs for service development and IoT device integration (M7- M36)*, that will enable development of services on top of the SocIoTal architecture, having in mind that the targeted users are public at large.

The API specification follows the best practices in specification of such interfaces. A REST style API is the envisioned approach (but not limited to) to be used, when suitable and convenient.

This deliverable, through its Section 1, proposes a semi-formal notation for the API specification description, which is carried out in Sections 2, 3 and 4, grouped by envisioned modules of the SocIoTal architecture and those provided by a selected base platform as prerequisite for SocIoTal.
Each API description includes the supported functionalities list (i.e., CRUD operations in case of a REST-based API), data types and their representation as well as returned data and parameters; moreover, they include the scope of the API, i.e.: internal or public, and the provider as the architecture module, WP or external tool/framework which expose the particular API.
The deliverable highlights two main aspects worth noting:

- The basic functionalities desired from a platform that must be the core module of the SocIoTal architecture and providing the lower-level infrastructure of an IoT platform and framework.
- The functionalities exposed by SocIoTal to support people who want to develop new applications based on it.

As a side-effect then this deliverable outlines several relevant modules of the system, the integration interfaces intended for an "internal" use inside the SocIoTal architecture and the potential of the platform toward third-party applications development.

The final version of the API along with the full documentation will be made public once the whole SocIoTal architecture will be delivered as stable.

Section 6 - References

[1] SocIoTal D2.1 - IoT Communities and Identity Management
[2] SocIoTal D4.1 - Feature specification of intuitive user and developer environment (Section. 2.4.2 Domain Model for SocIoTal UserEnv)
[3] IoT-A-D1.5 - Final architectural reference model for the IoT v3.0
[4] SocIoTal D1.2.1 – First version of SocIoTal architecture (Section 4.3.1)
[5] IoT-A D4.3 – Concepts and Solutions for Entity-based Discovery of IoT Resources and Managing their Dynamic Associations

Section 7 - Glossary

API: Application Programming Interface
CRUD: Set of operations for a resource, Create-Read-Update-Delete.
F2F: Face-to-face, referred to personal interactions that occur in proximity
HTTP REST: Hypertext Transfer Protocol Representational State Transfer
IoT: Internet of Things
IoT-A: an architectural reference model for IoT achieved by the IOT-A project
MQTT: Machine-to-Machine (M2M)/Internet of Things
P2P: Peer to peer
QR: Quick Response Code
URL: Uniform Resource Locator
UserEnv: The SocIoTal End User Environment
VE: see Virtual Entity
Virtual Entity: Computational or data element representing a Physical Entity in IOT-a model
WS: Web Service
WP: Work Package
XACML: eXtensible Access Control Markup Language
XML: eXtensible Markup Language
JSON: JavaScript Object Notation
SensorML: Sensor Model Language
GPS: Global Positioning System
GNSS: Global Navigation Satellite System
VE: Virtual Entity