"Linked Open Apps Ecosystem to open up innovation in smart cities"

Project Number: 297363

Deliverable:	**D4.5 SDK**
Version:	**1.6**
Delivery date:	**06/02/2013**
Dissemination level:	**PU**
Author:	**RETEVISION**

Statement of originality:

This deliverable contains original unpublished work except where clearly indicated otherwise.

Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Summary

The document presents the first version of the iCity SDK for each potential user who will develop an application to interact to iCity Platform.

The main objective of iCity SDK is to facilitate the creation of iCity Applications.

DOCUMENT HISTORY

Version	Date of issue	Status	Content and changes	Modified by
0.1	05/11/2012	Draft	Creation	Retevision
0.2	07/01/2013	Draft	Overview of SDK Design	Retevision
0.3	15/01/2013	Draft	Minor changes	Retevision
0.4	22/01/2013	Draft	iCity Service Certification Process defined	BCN
0.5	25/01/2013	Draft	SDK iCity Prototype defined	Retevision
0.6	06/02/2013	Draft	Final version peer review	Retevision

Document contributors

Partner	Contributor
Retevision	Carmen Vicente, Javier Marcos, Luis Moreno y Elena Villa
BCN	Marc Garriga and Miriam Alvarado

TABLE OF CONTENTS

TABLE OF FIGURES

Abbreviations and Acronyms

Acronym	Description
API	Application Programming Interface
DDD	Domain Driven Design
OGC	Open Geospatial Consortium
O&M	Observation and Measurements
SAS	Sensor Alert Service
SDK	Software Development Kit
SOS	Sensor Observation Service
TDD	Test Driven Development

1. Introduction

1.1 Purpose of this document

The aim of this document is to describe two topics; the first one is about the architectural overview and technical considerations to create a SDK prototype for iCity and the second one is about a set of design rules that will guide this prototype and its future versions.

Is remarkable to say that the ultimate objective of the strategy behind these topics is to spread the using of these services around the 'techies' segment, with a special focus in those involved in the development of mobile technologies, this does not means forgetting the traditional ones (as desktop or server applications) but to offer the right conditions to propagate its using in an environment which is more and more delocalized.

Finally, it is important to mention that this document will be alive and modified during the whole life of the project in order to adapt its contents to the final iCity platform architecture. It will gather in the following deliverable:

- D4.10 (M24).
- D4.15 (M35).

1.2 Objectives

So the main objectives of iCity SDK are the following:

- To give the project an appropriate access to the platform.
- To give the project an appropriate access to Cities infrastructures in a transparent way.
- To give support an application abstraction layer to easily integrate with other applications.
- To keep in touch with stakeholders in order to improve the functionalities of the iCity SDK.
- To provide different tools for developers to develop new apps.
- To develop an innovative method for user interactions, using different operating systems.

2. iCity mapping SDK

The following image shows the iCity platform prototype where SDK is the main interface between applications and iCity platform.

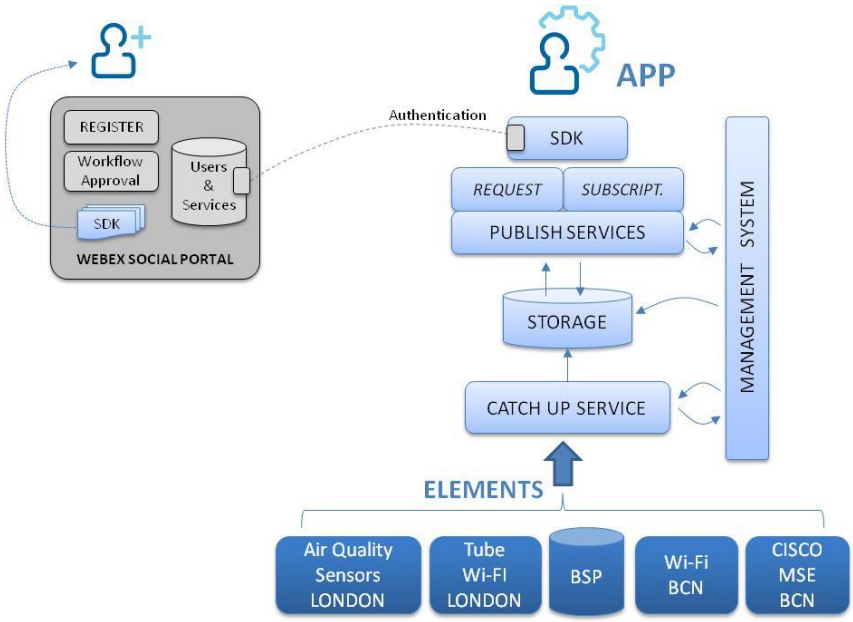

Figure 1: 1st version of iCity prototype

The prototype follows the architecture designed and defined by WP3. Figure 2 shows the fist prototype of SDK and their mapping over iCity architecture defined by WP3.

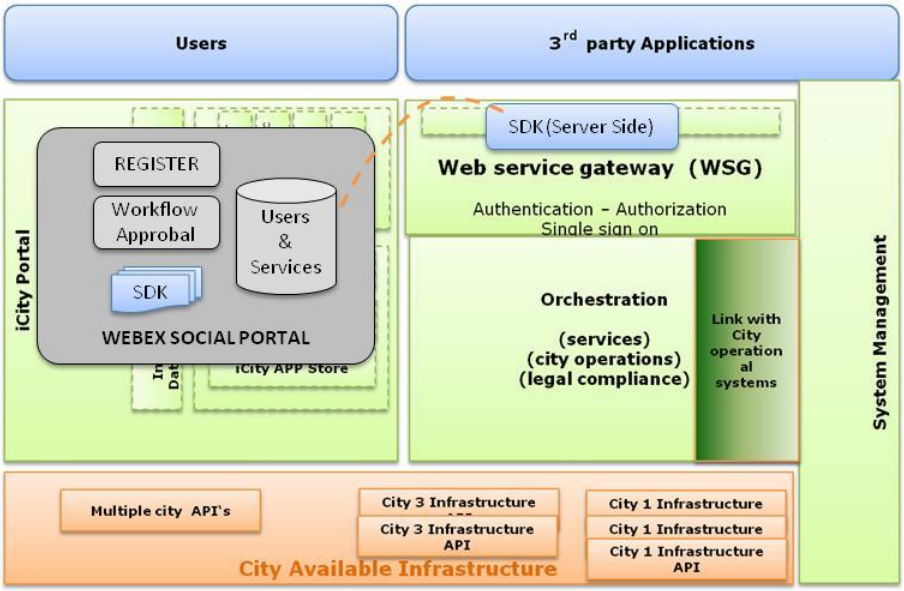

Figure 2: Mapping SDK over iCity architecture

For the correct operation of the SDK is needed the following modules in the iCity prototype:

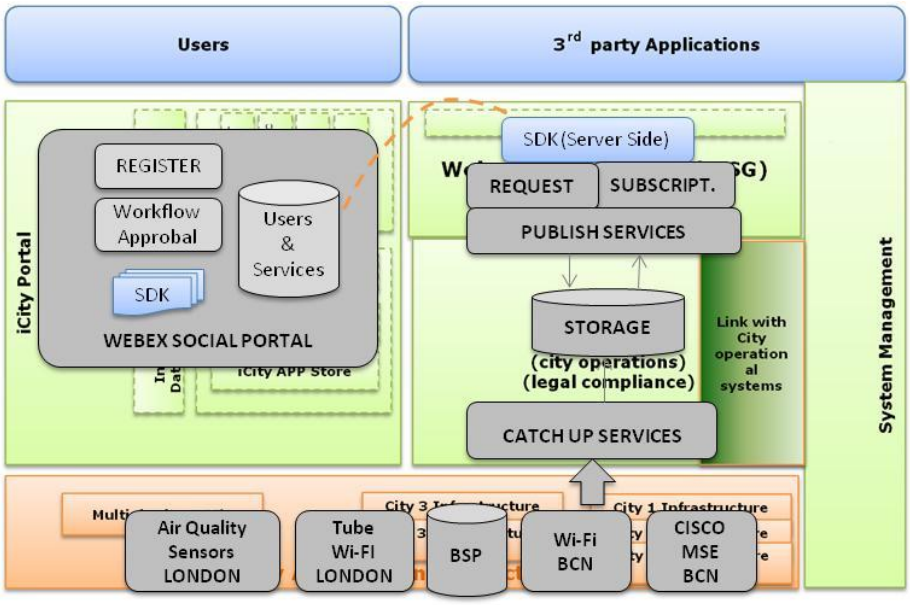

Figure 3: Mapping iCity Prototype over iCity architecture

The **catchup services** module provides the connection of both, the open infrastructures and 3rd parties' platforms, to iCity platform. It is formed by adapters, where every adapter interacts with a specific open infrastructure or platform. Thus, when a new infrastructure or platform will be opened to iCity platform, just a new adapter must be deployed to integrate the new API of the infrastructure. This implementation allows easy growing and also technological interoperability at this layer of the iCity platform.

The **storage module** of the prototype has been developed using a relational database. The data structure is based in OGC.

The **publish service** module is based on OGC[1] standard protocol and supports both types of publish services: request and subscription.

- **Request services** allow queries to obtain historical data and also actual data.
- **Subscription services** allow queries to subscribe to a type of data defined by a filter during the subscription process. Thus, queries will receive the data when obtained.

The iCity apps interact with the opened infrastructures through the iCity SDK, but not directly with the opened infrastructure. The publish service provide the data to SDK through Storage module. And even more important, the catchup services module transforms the data collected from heterogeneous iCity infrastructures and stores in a homogeny way.

Finally, SDK allows the implementation of security access to open infrastructures.

[1] Open Geospational Consortium

3. iCity SDK Workflow

3.1 Workflow for developers to access the portal

Webex social Portal allows developers registration process and provides through SDK a set of tools to help in the creation of applications. The following image shows the developer registration process.

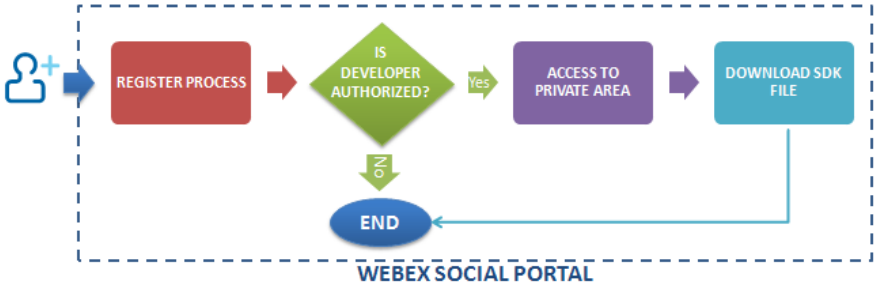

Figure 4: Workflow for developers to access the portal

3.2 Workflow for application to access the iCity platform

SDK provides an access to data for applications built by developers, whenever developers are authorized to access this data. The following image shows the application access.

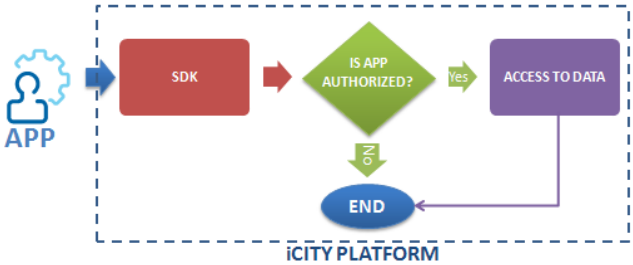

Figure 5: Workflow for application to access the iCity platform

4. iCity SDK Definition

In this chapter we are going to describe what a SDK means, which components does it contain and which is the relation between the SDK and the other components of the icity platform.

iCity Software Development Kit (SDK) provides all the functionalities and features needed to build powerful and innovative open infrastructure applications. The iCity SDK consists of a series of modules, or building blocks, easily configurable and extendable. The modules address different kind of functionalities and are grouped into Client, server and communication. They are flexible and could be adapted according to costumer needs.

Consequently with the strategy, the proposed architecture, platform and design rules will be focused on guarantee the next characteristics (sometimes opposite by their natural trend) that we will define as the Design Principles:

- **Ease of use**,
- **Compatibility**,
- **Extensibility**,
- **Portability** and**,**
- **Maintainability.**

Ease of use: this means the SDK has to encapsulate the Domain Model and make the services transparent to the client. It also means helpful documentation and samples to assist in its using.

Compatibility: Nonetheless, despite the encapsulation the SDK has to guarantee to keep providing the raw xml information as it comes from the services and sensors, in order to allow new definitions (or not defined in the current SDK version) that client can handle by herself while the SDK does not encapsulate it.

Extensibility: The SDK should provide Interfaces and abstract entities that can be easily extended by the client, to create their own models (still keeping compatibility).

Portability: The SDK should use a standard that gives flexibility in order to allow its using between different platforms. The language chosen for the prototype and the recommendation for other SDKs development is C# in combination with .NET and Mono compilers that can make the SDK's assemblies reach maximum market quota

Maintainability: The SDK should be developed according to a modern architecture pattern, this includes following DDD, TDD and N-Layers. In the user's side, it should provide a web environment where to place the information, FAQ, bugs and issue tracking. Finally it should consider a marketing strategy that includes blog, events and community participation.

5. iCity SDK Design

5.1 Comparison between WS, API or SDK

Although it is out of the scope of this paper to give a fully description of each of these terms, because this definition is done in WP3, it is necessary to expose them and the relation they keep together, so that clarify the reason of use a SDK instead of a Web Service or API.

According to the picture below we can see that in the current context they all complementary and subsystem for the system above, being the **Web Service** the communication channel, the **API** the client assembly that encapsulates the model and exposes/consumes the service, and the **SDK** as the previous two ones plus the set of tools, information, samples, tutorial and whatever other resource that is provided to the developers.

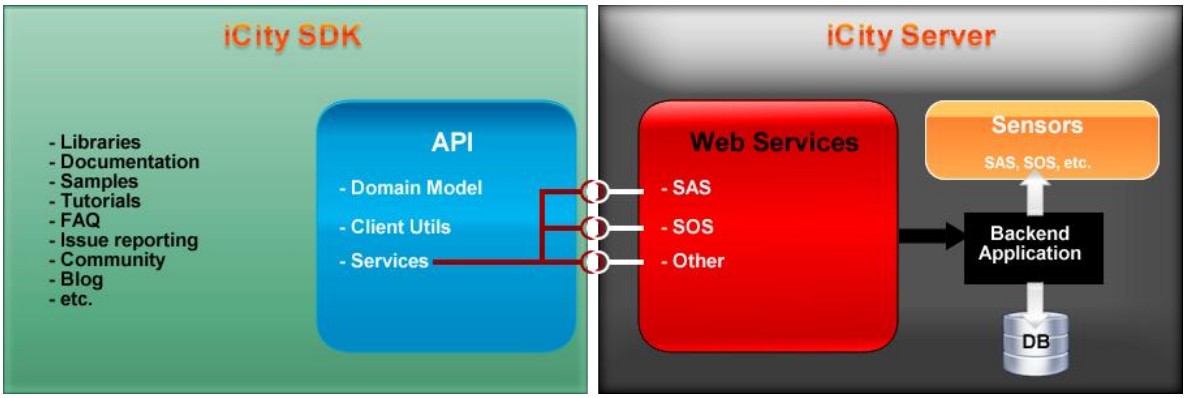

Figure 6: Workflow for application to access the iCity platform

In order to be as generic as possible, we have decided to use an SDK based on the OGC standard Sensor Web Enablement (SWE). The Open Geospatial Consortium (OGC) was established in 1994 and it's composed by both many public and private organizations. Its main purpose is to define open standards and interoperable within the GIS and the World Wide Web. They pursue agreements between different companies that enable the interoperation of geoprocessing systems and facilitate the exchange of geographical information.

The Open Geospatial Consortium's Sensor Web Enablement (SWE) activities, which have been executed principally through the OGC Web Services (OWS) initiatives under the Interoperability Program, is establishing the interfaces and protocols that will enable "Sensor Webs" through which applications and services will be able to access sensors of all types over networks such as the Internet and with the same standard technologies and protocols that enable the Web. These initiatives have defined, prototyped and tested several foundational components needed for a Sensor Web, namely:

1. **Observations & Measurements (O&M)** - The general models and XML encodings for sensor observations and measurements.

2. **Sensor Alert Service (SAS)** – A service by which a client can register for and receive sensor alert messages. The service supports both pre-defined and custom alerts and covers the process of alert publication, subscription, and notification.
3. **Sensor Model Language (SensorML)** – The general models and XML schema for describing sensors and processes associated with measurement.
4. **Sensor Planning Service (SPS)** – A service by which a client can determine collection feasibility for a desired set of collection requests for one or more sensors/platforms, or a client may submit collection requests directly to these sensors/platforms.
5. **Transducer Markup Language (TML)** – General characterizations of transducers (both receivers and transmitters), their data, how that data is generated, the phenomenon being measured by or produced by transducers, transporting the data, and any and all support data (metadata) necessary for later processing and understanding of the transducer data.
6. **Web Notification Service (WNS)** – A service by which a client may conduct synchronous dialogues (message interchanges) with one or more other services. This service is useful when many collaborating services are required to satisfy a client request, and/or when significant delays are involved is satisfying the request.

A **Sensor Observation Service** provides an API for managing deployed sensors and retrieving sensor data and specifically "observation" data. Whether from in-situ sensors (e.g., water monitoring) or dynamic sensors (e.g., satellite imaging), measurements made from sensor systems contribute most of the geospatial data by volume used in geospatial systems today.
The goal of SOS is to provide access to observations from sensors and sensor systems in a standard way that is consistent for all sensor systems including remote, in-situ, fixed and mobile sensors. This is a challenging task because the users of sensor data have historically been divided into those who primarily deal with in-situ sensors and those who primarily deal with remote sensors. The terminology, perspective, and expectations of these two broad groups are different. SOS leverages the Observation and Measurements (O&M) specification for modeling sensor observations and the TransducerML and SensorML specifications for modeling sensors and sensor systems.

The approach that has been taken in the development of SOS, and the SWE specifications on which it depends, is to carefully model sensors, sensor systems, and observations in such a way that the model covers all varieties of sensors and supports the requirements of all users of sensor data. SOS leverages the standard properties of these two data types (sensors, observations) to provide specialized operation signatures for observation data.

SOS has three mandatory "core" operations: **GetObservation, DescribeSensor**, and **GetCapabilities**. The **GetObservation** operation provides access to sensor observations and measurement data via a spatio-temporal query that can be filtered by phenomena. The **DescribeSensor** operation retrieves detailed information about the sensors and processes generating those measurements. The **GetCapabilities** operation provides the means to access SOS service metadata. Several optional, non-mandatory operations have also been defined. There are two operations to support transactions, **RegisterSensor** and **InsertObservation**, and six enhanced operations, including GetResult,

GetFeatureOfInterest, GetFeatureOfInterestTime, DescribeFeatureOfInterest, DescribeObservationType, and DescribeResultModel.

Used in conjunction with other OGC specifications, the SOS provides a broad range of interoperable capability for discovering, binding to and interrogating individual sensors, sensor platforms, or networked constellations of sensors in real-time, archived or simulated environments.

From now, in the iCity case, we use the "core" operations, in order to obtain all the information related to the open infrastructures involved in the iCity Project.

Get Capabilities:

This function allows obtaining the main information about the system which could be divided in two areas:

- System information
- System offerings

We mean by system information, the name and description of the service, version and people involved in the project and all the methods supported by the standard.

We understand by system offerings, all the devices of the system, its positioning, measures and the response format.

At the beginning we could use this function in order to know all the capabilities of the system.

DescribeSensor

This function allows obtaining a complete description of a single device of the system (sensor, hub, etc..).

It includes this information:

- Identifier: identifies the device with a unique identifier.
- Description: describes the element.
- Position: position of the element.
- Information type: manufacturer, version, etc…
- Dependencies: in case the element pertains to another one.
- Measures: information about the measures provided by an element

5.2 Architecture

According with the Design Principles we will split the architecture in three different scopes:

- **Common Library**

  It keeps the Domain Models described in the OGC[2] encapsulated as well as the services contracts and other common utilities.

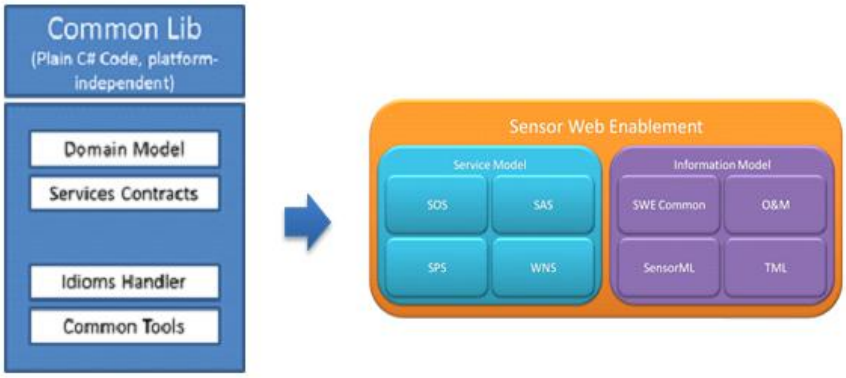

Figure 7: Common Library

- **Client API**

This encapsulates the common lib and provides it as an assembly compiled for the supported platforms that will be consumed by a client application. The main functions of this API are to make the using of the services transparent to the user and to offer extra functionality which can be common for every platform (conversions, format, etc) or specific behaviour for each one (e.g. special classes or properties for iOS, Android, Windows, etc.).

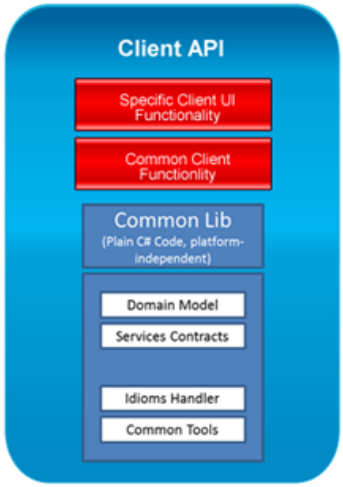

Figure 8: Client API

- **Server Services**

[2] Open Geospational Consortium

This scope involves all the aspects related to adapt the server to provide the services according with the Services Contracts defined in the Common Library. If the server is created from the raw it should follow the Design Principles implementing not only the Common Library but following also DDD, TDD and N-Layers methodologies.

It is implementation decision to decide channels and communication technologies, in fact there are plenty of alternatives that are good candidates as it shown in the picture below, but it is mandatory that both the API and the Server implements the same Services Contracts with the same behaviour so the communication will be transparent and will affect exactly equal to every client no matter the platform.

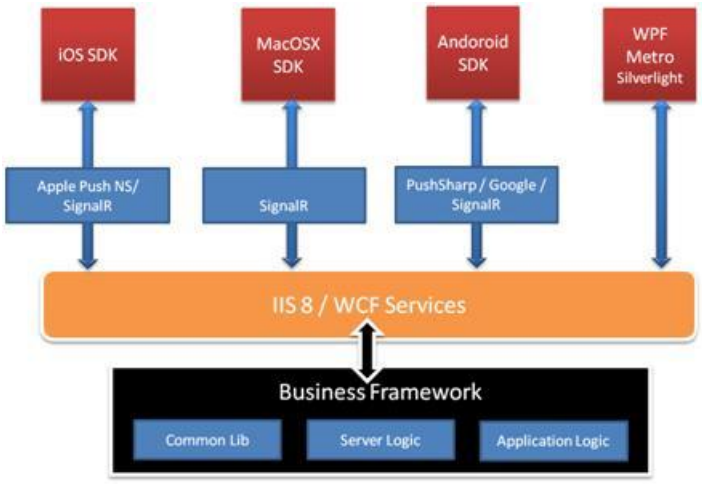

Figure 9: Server Services

5.3 Modules

We will classify the modules according with the scope define before:

Common Library

- **Domain Model**: This domain model is subdivided also into two different parts:
    - **Metadata and System models**: the properties and entities described in the O&M (e.g. Observation, Process, Feature Type, etc.) and that are used generically by all the services when providing actions (e.g. get capabilities, subscribe, etc.) and information coming from the sensors or alerts.
    - **Sensor models**: unities and data types that are provided by supported sensors and alerts by a service. This also includes conversion entities according with the unity types.
- **Services Contracts**: The service contracts are the signature of the methods provided by each service exposed and define specific (and common between the server and the client API) input and output types. There are also two different parts:

  - **Operational Services Contracts**: those ones that correspond to the available services (e.g. SOS, SAS, etc.) There will be one Service Contract file for each one of the services.
  - **System Services Contracts**: those ones that are not included in the previous one, usually Permissions, Registration, Idioms, etc. Usually those related to administrative actions.

- **Idioms Handler**: In this module is placed everything related with Localization Resources (dictionaries, languages available, etc.) - but the Service Contracts - in order to provide culture adaption capabilities to the clients (in case they need it).
- **Common Tools**: Common libraries of programming tools that can be shared between Server and clients

Client API

- **Common Library**: described before but mentioned here as it is the main part of the API core.
- **Common UI functionality**: This component is optional and it depends on the functionality strategy. This would include View Controllers (Presenters, View Models, etc.) non-platform dependent with the purpose to provide predefined views to clients. They must be independent of platform and expose extensibility for platform specific actions.
- **Common functionality**: Those programming tools that can be share between the different platforms but not with the server (in other words, every client-side library tool that cannot be placed in the Common Library)
- **Platform functionality**: In case it is necessary, before compiling the API for each different supported platform, it will be added to an intermediate native project that will extend the API with platform specific actions.

Server Services:

- **Common Library**: described before but mentioned here as it is necessary to construct and to expose the services that will be consumed by the Client API.
- **Service handler Layer**: this is the intermediate layer between the Server backend application and the services exposed, in case they already existed before and not created based on the Common Library here it will be also a translation component between the existing server Domain Model and the Common Library one.
- **Service Layer**: Exposed through interfaces, here will be placed each one of the services defined in Common Library Contracts. It's up to the implementation to decide if the services will be exposed directly (for instance in case of it is still wanted that the services are able to be consumed as web services without the SDK) or

through a proxy service that then redistribute the service internally to the service handler layer.

6. iCity Service Certification Process

Creating a service in the iCity Platform probably has some trust, technical and even legal consequences. So, a certification process must be mandatory in the iCity project.

Before a service is used, we need to verify it's compliant with the iCity platform rules, city strategy, legal and also technical aspects, this is a critical part of the iCity platform.

The purpose of this paper is to explain this certification process for services that use open infrastructures within the project iCity.

This process (it is mandatory and free of cost) has 3 main approval layers: city strategy, legal aspects and technical questions.

6.1 The petition

A third party organization makes a petition in order to create a new service using iCity platform. This petition contains meta-information about the service proposal, in fact, every iCity app is defined by a set of metadata. This information will be provided by the iCity Service developer when starting the iCity Service Certification Process and reviewed later if necessary. This is important because before a service is published, has to verify that it is compliant with the iCity platform rules in three main areas: city strategy, legal aspects and also technical questions.

The meta-information of each iCity app should be (some of this information will be available at the beginning of the certification process, other will be at the end of this process or when the service is published).

- Service name.
- Short description.
- Size (in Kb).
- Images (logo + screen captures).
- Owner/Developer.
- Last version available, Date.
- Language.
- User license.
- Which infrastructures are to be used?
- Cost, if any? If so, what does it cost?
- Final devices:
    - Mobile: which OS? Google Android, Apple iOS, Symbian, Microsoft Windows Phone,etc…
    - Web: Supported browsers
    - Tablet

- To which city is it aimed for?
- Mapping (entire city, a district, a particular neighbourhood, a street, a particular point, etc).
- Which is the topic?
- QR code or Link to directly download the result application.
- Score that users give*.
- Users' comments*.

* These two blocks of information are results of the natural use of the service and feed by users, so will be a part of the service's metadata when it is available in the iCity Platform.

6.2 City Strategy Approval

The city (where this new service will run) has to check if this new service "matches" the city strategy. If the result is "not valid" then the third party has to modify the petition.

This City Check has three steps:

1. First, the officials of the City must validate the petition; this is a "simple" check in order to ensure that the City has all data needed of the petition. As a result of this step, the City will know which infrastructure is involved in the petition.

2. The second step is the strategic validation of the petition. A high-level team of the City must decide if this petition is according of the City strategy.

3. When the high-level team validates the petition, then the infrastructure owner must analyse if the petition matches the technical limits of the infrastructure. In addition, the owner has to estimate the cost of the eventual changes to do in the infrastructure.

6.3 Legal Aspects Approval

Obviously, the petition has to be law-abiding. So, in parallel of the city strategy checkpoint, the service will be tested in legally aspects. Again, if the result is "not valid" then the third party has to modify the petition in order to make it fit to the legal framework.

6.4 Technical Aspects Approval

This step could have several "loops". The first one is to ensure that the petition "matches" the technical aspects prior to development. If the result is "not valid", then the third party has to modify the petition only the technical aspects. If the modification includes more than technical aspects, then the process goes back to start point. At this point, the third party can start the developing phase.

When the service past the last loop of the technical approval, then the petition is a real development, it has passed the technical approval. Automatically it will be added in the iCity Apps Store (according to meta-information of the petition).

This is the basic definition of the certification process for a new service, in the case of an upgrade of pre-existing service, then could be only a single loop of technical approval or all process, it depends on how deep the changes are.

iCity Service certification process schema:

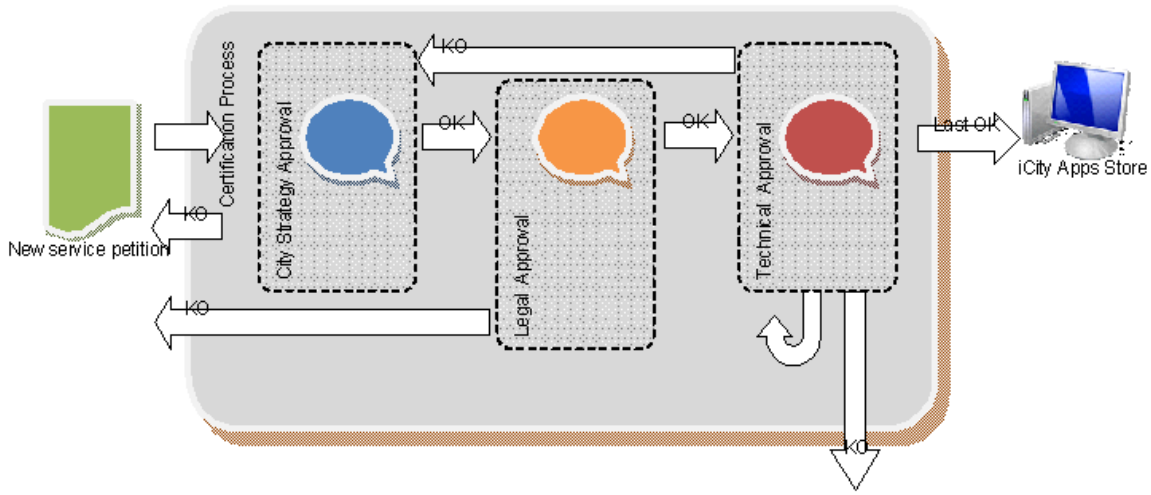

Figure 10: iCity Service certification process schema

7. iCity SDK Prototype

According with the previous points definitions, we are going to define the structure that will support the iCity SDK. This index is classified according with a different scope that the one defined in the design guide as it is now understood that the common library is included within the Client API and Server Services ones. Therefore, the iCity SDK will have three different modules:

- Server side
- Client side (web site based)
- Collaborative tool (web site based)

7.1 Server side

There are two different functions that have relation with the iCity SDK:

- One that provides the Services that are going to be consumed by the SDK
- One that provides hostage for the Client and the Community sides project parts.

The scope of the Server side topic is to describe the first one while the web hostage of the SDK will be defined in a different project.

7.1.1 Platform Overview

It is up to the implementation to decide if a new server application (full functional[3]) will be created, an existing one will modify or a new mirror server will be placed[4]. Whatever the server application strategy is decided it will expose Services Contracts defined in the Common Library and will present the topology described in the picture below (exactly the same described during the Design):

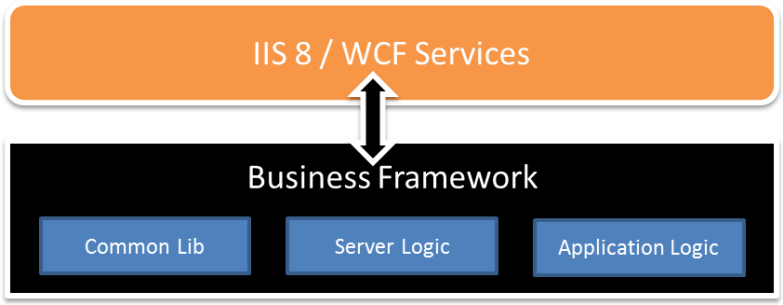

Figure 11: Platform Overview

7.1.2 Services Protocols

- Operation Services Contracts (SOS,SAS)

3 This means developing the whole Framework for the application to handle the domain, persistence, authentication and all the other server functionalities.
4 Consuming services from the first one and redirecting it to the SDK.

- System Services Contracts

We will not extend in the Operational Services except for saying that current scope of iCity is exposing SOS and SAS for more details about them please go to the referenced documentation.

About the System Services Contracts they will include:

- User authentication
- Idioms Services
- Incidence tracking

For the prototype:

- Operation Services Contracts (SOS, SAS): all the services will be exposed but only a functional subset of the Domain Model will be provided (to be defined in the Prototype Specifications).
- System Services Contracts: simplified versions of the services mentioned above (to be defined in the Prototype Specifications).

7.2 Client side

This section corresponds to the set of tools that we provide to developers through collaboration tools. The index of contents it will present in the following charters.

7.2.1 Download Section

- Libraries (SOS API, SAS API, OGC Model for SAS & SOS)
  - Windows 7/8.
  - Windows Phone 7/8.
  - Android.
  - iOS.
  - Linux.
  - Other.
- Installation Guide.
- Requirements & Compatibility.

For the prototype:

- Libraries:
  - Windows 7/8.
- Included in the prototype a simplified installation guide.
- Included in the prototype a simplified requirement & compatibility document.

7.2.2 Documentation

- Platform Overview
- Configuration & Connectivity
- Components
  - SOS API part (To do: Identifiy I/O parameters and extra methods)
  - SAS API part (To do: Identifiy I/O parameters and extra methods)
  - OGC Model (To do: Identify domain entities, create base entities)
- Extensibility Guide (eg Sensor ML, SPS, etc)

7.2.3 Samples

There will be samples for each of the main platforms and they will cover different functionality provided by the different Services exposed and the UI extensions (common and platform-specific).

For the prototype is given a sample about SOS service.

7.3 Collaborative tool

Collaborative tool is a web site environment that integrates the following set of community applications useful for developers, for example:

- Forum.
- FAQ.
- Wiki.
- Blog
- Instant message service.

For the prototype is integrated the Webex Social Portal which provide some of the functionalities listed above.

7.3.1 Developer registration process

Web site environment tool provide a registration process. Once developers have a successful registration, they will be able to download a client SDK.

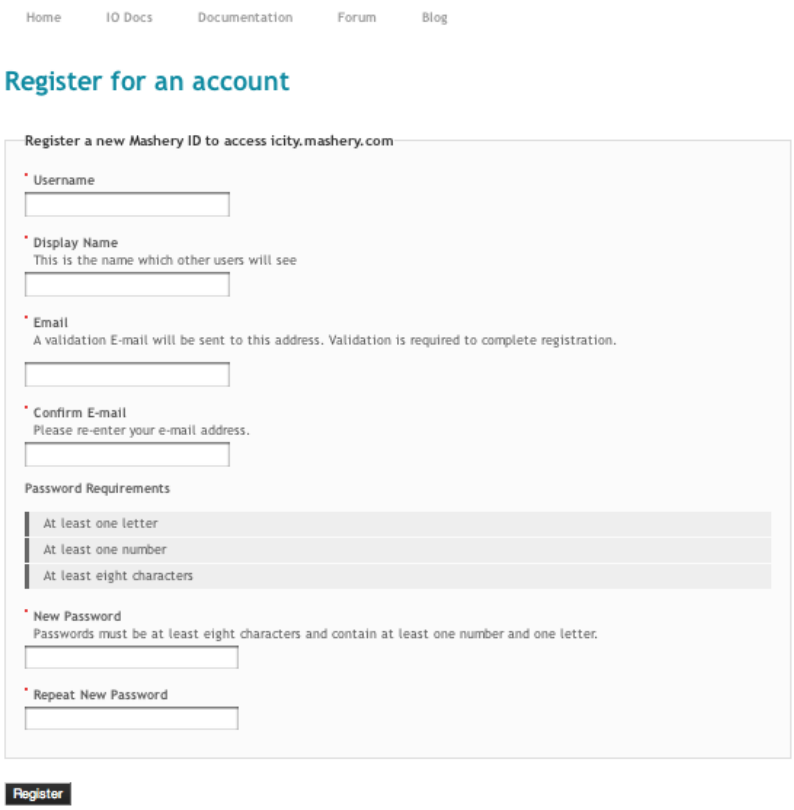

Figure 12: Developer registration Process

7.3.2 Developer Forum

Developer forum provides an easy way to communicate with different developers to explain the doubts and knowledge related of the SDK client.

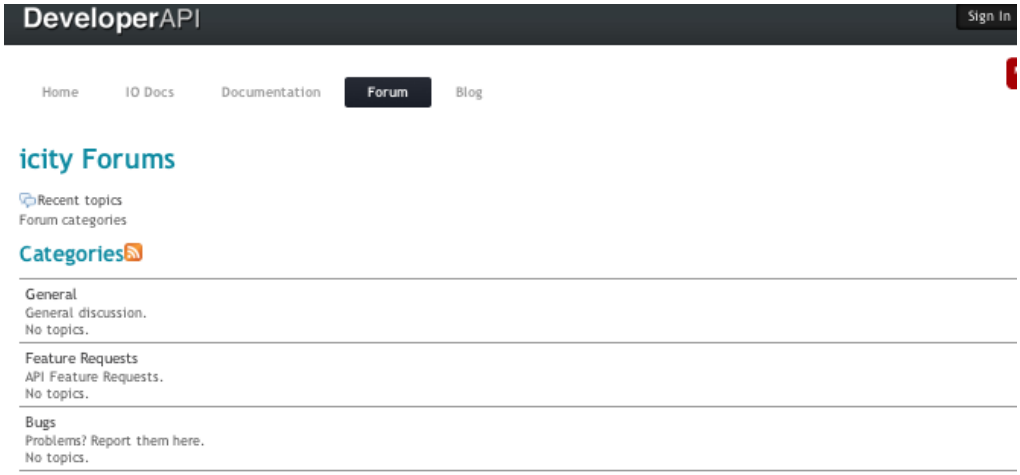

Figure 13: Developer Forum

7.3.3 Developer documentation

Webex Social portal provide a space where the community can share documents and also they can find document related a specifications of SDK.

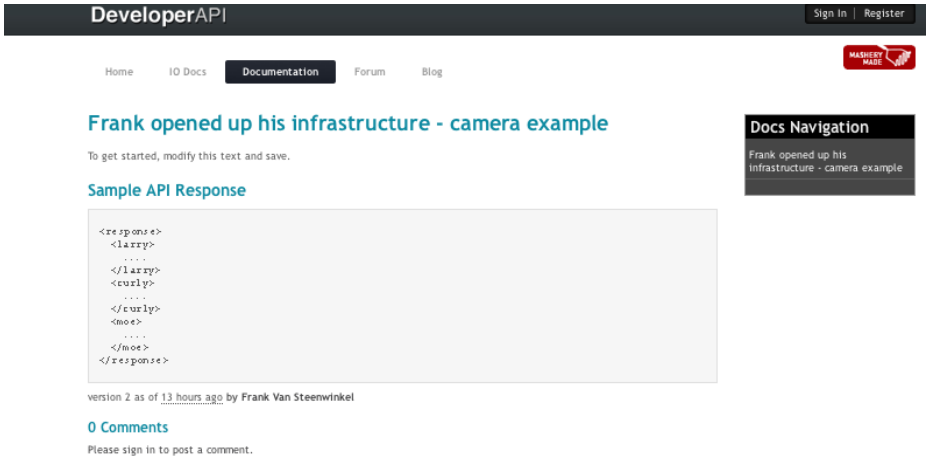

Figure 14: Developer documentation

8. Conclusions

During the first year of iCity project, WP4 has developed a set of tools that allows standard access to iCity Platform for any developer who wants to develop an iCity application. Once an application is created and iCity certified, the application can consume the services offered by iCity platform through the SDK, which provides a security access to the iCity services.

When developing using the iCity SDK, the key benefits are:

- Reduced time-to-marked- reuse the complete Library for creating your applications.
- One (or as few as possible) code base(s) for all platforms and devices.
- End-to-end solution: server framework, communication layer and client framework.
- Network, content, device, and OS agnostic.
- Secure communication protocol.
- Easy operation and distribution of your application.

Finally, the iCity SDK would help the engagement community process as it is required by WP2 and WP5.