DELIVERABLE

Project Acronym:	**CitySDK**
Grant Agreement number:	**297220**
Project Title:	**Smart City Service Development Kit and its Application Pilots**

Deliverable 2.2: Progress report, Pilots and SDK

Revision: 3

Authors:

Geert Monsieur (UVT)

With contributions from:

Hanna Niemi-Hugaerts (Forum Virium)
Rajaniemi Jaakko (Helsinki)
Job Spierings (Waag)
Tom Demeyer (Waag)
Ricardo Lopes Pereira (IST)
André Oliveira (ISA)
Valter Ferreira (CML)
Nuno Xavier (CML)
Jean Barroca (Alfamicro)

Project co-funded by the European Commission within the ICT Policy Support Programme		
Dissemination Level		
P	Public	X
C	Confidential, only for members of the consortium and the Commission Services	

REVISION HISTORY AND STATEMENT OF ORIGINALITY

Revision History

Revision	Date	Author	Organisation	Description
1	1/5/2013	Geert Monsieur	UVT	Initial verion
2	24/6/2013	Geert Monsieur	UVT	Participation and Mobility pilot preparations added
3	28/6/2013	Geert Monsieur	UVT	Tourism pilot preparations added

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This deliverable contains original unpublished work except where clearly indicated otherwise.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

1 Introduction

In preparation for the CitySDK pilots it is crucial that all project partners share a similar vision on what the future CitySDK ecosystem should look like. Therefore, in Section 2 we have described this shared vision. Moreover, in order to support this ecosystem it is also important to strive for an uniform design and documentation on the CitySDK APIs, across the three functional domains (participation, mobility and tourism). Guidelines on the further design changes and the API documentation are presented in Section 3. Domain-specific information on the pilot preparations are given in Sections 4 to 6.

2 CitySDK ecosystem

One of the most important goals in CitySDK is to create an **ecosystem** that facilitates the work of an app developer by having unified and open city interfaces available across different cities in Europe. While CitySDK is organized in three functional domains (participation, mobility and tourism), such an ecosystem also needs to stimulate the creation of **innovative cross-domain apps** (e.g. participation-like reporting of an issue with a bus stop known in the mobility domain that is close to a museum known in the tourism domain).

2.1 Current CitySDK APIs

2.1.1 Participation

Through the *CitySDK participation API* one can post and get **issues** that a citizen (or perhaps a tourist or businessman) would like to report to a specific municipality. Typically, such an issue (e.g. a street pothole) is related to one or more **locations**[1] in a city.

2.1.2 Mobility

The *CitySDK mobility API* currently only allows to retrieve (mobility) data. Writing data through the interface (e.g. for sharing experiences about public transport) is going to be added in the near future. The data that can be retrieved through the CitySDK mobility API are **nodes** and information related to these nodes. The concept of a node is based on the similar core element in OpenStreetMap (OSM). In practice, a node can be anything: a landmark, bus stop, stretch of road, a town or a public transport line. Essentially, every node consists of a **geospatial point, region or line**. The first-class nodes retrievable through the CitySDK mobility interface are related to public transport (e.g. GTFS-based data on bus stops, tram lines, etc.) and administrative regions (e.g. city boundaries). Additionally, any node available in OSM (e.g. a road, a museum, etc.) can be

[1] Adding a location to an issue is not mandatory, but implicitly an issue is always related to one municipality.

retrieved.
Each type of node is defined in a **layer** (e.g. the GTFS or OSM layer). The layers provide the real data. The main requirement for such a layer is the need to be able to define imported data as nodes (i.e. data need to be related to a specific point, region or line). The current implementation of the CitySDK mobility interface demonstrates that a broad range of data sets can be included as specific layers. More specifically, Waag's implementation includes layers for traffic information, rain forecasts and city statistics (e.g. population numbers).
The **value** in this approach lies in the fact that the concept of nodes is extremely simple and that it is complemented with the concept of layers. Since all nodes, whatever the type of node (i.e. the layer in which it is defined), are geospatially defined, it becomes possible to query data across layers. For example, the current CitySDK mobility interface allows to easily search for public transport lines in a certain (administrative) region; or find museums nearby a bus stop. In fact, the interface allows to execute almost every possible query in which you look for nodes (available in one or more layers) that have a certain geospatial relationship (e.g. bounding box, radius or administrative region) to other nodes (possibly in another layer).

2.1.3 Tourism

The data available through the current *CitySDK tourism interface* are fundamentally based on Points of Interest (POIs). Tourist **events** (e.g. a temporary exhibition in a museum), tourist **places** (e.g. hotels, museums, restaurants, etc.) or tourist **routes** (i.e. a route by a set of tourist places) are typically related to one or more **locations** in a city.

2.2 How to stimulate the innovation in the ecosystem?

Given the existing CitySDK APIs today an important question remains: How can we organize the CitySDK interfaces in a way that it is convenient for the developer and stimulates innovation?
In the context described above, there are many *forces* that make it hard to come up with an answer to this question. Any proper solution should balance these forces as much as possible.

- Implementing one integrated cross-domain API might be more complex than implementing three separate domain-specific interfaces.
- An integrated API stimulates developers to use and combine data from different domains, which might facilitate the development of new innovative cross-domain apps.
- Reaching a domain-specific audience might be easier when you present a domain-specific API (e.g. like the current CitySDK interfaces).
- In all three domains it should be possible to query data geospatially[2]. Having three different domain-specific APIs with possibly different ways to construct such queries can be confusing for a developer.

[2] Although it is not mandatory to add locations in issues reported through the participation service, we expect that the majority of issues have a location associated and hence can be integrated in a cross-domain interface that is founded on nodes/locations.

- Having uniform APIs in all three domains (e.g. all domains share the node concept and the way how nodes can be queried) might result into less efficient communication to developers, when compared to one integrated cross-domain interface.
- Changing CitySDK goals in a way that only one integrated cross-domain API is developed is extremely challenging for all project partners.

2.3 Proposed solution

Basically, there are three possible solutions for organizing the CitySDK APIs:

- Three separate CitySDK APIs, one for each domain.
- One integrated cross-domain API, in line with the current CitySDK mobility API that is fundamentally based on nodes and layers.
- **One integrated cross-domain API that is implemented using three separate CitySDK APIs, one for each domain.**

Based on the forces described above, we propose to realize the third solution. This solution is relatively easy to implement as it keeps the three separate domain-specific APIs[3]. At the same time, it includes an integrated cross-domain API that should bring important advantages for CitySDK (i.e. stimulate the development of innovative cross-domain apps).

3 Uniform Design & Documentation across domains

3.1 Uniform API Design

In this document we would like to list up best practices in (REST) API design that are important for CitySDK in order to obtain *uniform* interfaces across all domains.

DESIGN PRACTICE 1

Use **forward slashes** ('/') in the path portion of the URI to indicate a **hierarchical relationship between resources**.

e.g.: http://api.canvas.restapi.org/shapes/polygons/quadrilaterals/squares

DESIGN PRACTICE 2

Use **identity-based values in variable URI path segments.**
Some URI path segments are static; meaning they have fixed names that may be chosen by the REST API's designer. Other URI path segments are variable, which means that they are automatically filled in with some identifier that may help provide the URI with its uniqueness. The URI Template syntax allows designers to clearly name both the static and variable segments. A URI template includes variables that must be substituted before resolution. The URI template

[3] In practice, the integrated interfaces are going to be built on only two underlying interfaces (i.e. participation and tourism) as the CitySDK mobility interface already natively supports the idea of nodes and layers.

example below has three variables (leagueId, teamId, and playerId):
http://api.soccer.restapi.org/leagues/{leagueId}/teams/{teamId}/players/{playerId}
Each substitution of a URI template's variables may use a numeric or alphanumeric identifier, as shown in the examples below:
http://api.soccer.restapi.org/leagues/seattle/teams/trebuchet/players/21
http://api.soccer.restapi.org/games/3fd65a60-cb8b-11e0-9572-0800200c9a66

DESIGN PRACTICE 3

Many popular APIs (e.g. foursquare) use **JSON as the default format**. Since all current CitySDK APIs are already supporting JSON representations, CitySDK can follow a similar default approach. To request other content types in calls to CitySDK services, there are three main approaches, in order of preference:

- Using the **Content-Type HTTP header**
  - e.g. `Content-type: application/xml`
  - This keeps the URIs relatively simple, which should facilitate the use of the APIs.
  - Common content types can be found on http://www.iana.org/assignments/media-types
- Using a **parameter** named '**format**' in the URI
  - e.g. …/admr.nl.utrecht/nodes?format=json&osm::tourism=museum
  - This approach is easy to understand and it makes it easy to test and play with different formats in a browser.
  - One resource with different format representations has still one unique URI:
    - http://api.citysdk.waag.org/gtfs.stop.060671?format=json
    - http://api.citysdk.waag.org/gtfs.stop.060671?format=xml
- Using a sort of '**file extension**'.
  - e.g. http://api.citysdk.eu/pois/funnyPOIid123.json
  - This approach is easy to understand and it makes it easy to test and play with different formats in a browser.
  - More than one URI for one resource:
    - http://api.citysdk.eu/pois/funnyPOIid123.json
    - http://api.citysdk.eu/pois/funnyPOIid123.xml
  - This approach is used in the Open311 specification.

DESIGN PRACTICE 4

To limit the results that CitySDK services return we suggest a **pagination approach using** a **limit and offset** variable (e.g. as used in foursquare, box.com and Facebook), which is well known in databases and easy for developers.
e.g. http://api.citysdk.eu/admr.nl.utrecht/nodes?osm::tourism=museum&limit=5&offset=2

- The number of results returned is limited to five (limit=5) – results are returned starting from the third available result / the two first results are 'ignored' (offset=2).

By default, limit has the value 10 and offset is set to 0.

DESIGN PRACTICE 5

Since we are already sharing the first versions of our APIs with the developers, it is convenient to include a **version number in URIs**. Each URI should look like:
http://api.citysdk.eu/v[versionNumber]/funnyResource
(e.g. http://api.citysdk.eu/v1/ pois/funnyPOIid123)

DESIGN PRACTICE 6

When an error occurs a REST service typically returns a specific error code explaining what went wrong. To make life easier for a developer it is helpful to **align error codes to HTTP Status codes**.
There are five main categories of error codes:

- 1xx: Informational: Communicates transfer protocol-level information.
- 2xx: Success: Indicates that the client's request was accepted successfully.
- 3xx: Redirection: Indicates that the client must take some additional action in order to complete their request.
- 4xx: Client Error: This category of error status codes points the finger at clients.
- 5xx: Server Error: The server takes responsibility for these error status codes.

Commonly used error codes can be found on: http://www.restapitutorial.com/httpstatuscodes.html

DESIGN PRACTICE 7

The data that is provided through the CitySDK services can have different lifetimes. Therefore, it is convenient to use the cache-related headers in successful response messages (2xx):

- The **Last-Modified header** is a timestamp that indicates the last time that something happened to alter the representational state of the resource. Clients and cache intermediaries may rely on this header to determine the freshness of their local copies of a resource's state representation. This header should always be supplied in response to GET requests.
    - For example:
        - `Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT`
- A **Cache-Control header** in response messages with a max-age value (in seconds) equal to the freshness lifetime.
    - For example:
        - `Cache-Control: max-age=60, must-revalidate`
- To support legacy HTTP 1.0 caches, a REST API should include an **Expires header** with the expiration date-time. The value is a time at which the API generated the representation plus the freshness lifetime. REST APIs should also include **a Date header** with a date-time of the time at which the API returned the response. Including this header helps clients compute the freshness lifetime as the difference between the values of the Expires and Date headers.
    - For example:
        - `Date: Tue, 15 Nov 1994 08:12:31 GMT`
        - `Expires: Thu, 01 Dec 1994 16:00:00 GMT`

DESIGN PRACTICE 8

If a client wants to retrieve contents (e.g. museum description) in a specific language, it can specify the desired language by setting the **Accept-Language HTTP header** in the request. Alternatively,

you can add a locale=XXX parameter to your request but HTTP header specification is preferred. In the participation domain the **locale parameter** is currently used to indicate the language that the reporting citizen would like to use when interacting with the municipality. This needs to be changed (e.g. preferred_language) to avoid confusion with the locale parameter explained above.

3.2 Uniform API documentation

This section presents a proposal on what information the CitySDK API documentation web pages should contain. Any proposal on how to format these web pages is intentionally left out. Although consistent and well-thought formatting of the documentation can certainly improve the readability, the aim of this section is to focus solely on the *contents*.

The CitySDK API documentation has three main parts.

PART 1. General information about API:

- Base URI (see telco562013results in WP2 folder on SugarSync)
  (City-specific differences should be mentioned!)
  - Including version numbers (see document about design practices)
- Pagination (see document about design practices)
- Content type (see document about design practices)
- Licensing and keys (City-specific differences should be mentioned!)

PART 2. Resource-oriented presentation

- Overview web page:
  - Listing all resources, including hierarchical relationship between resources.
    For example:
    - /shapes
    - /shapes/polygons
    - /shapes/polygons/quadrilaterals
    - /shapes/polygons/quadrilaterals/squares
  - For each resource short textual description that explains what the resource is about and what you can do with it (e.g. query, create new resource, etc.)
  - Each resource is linked to the resource webpage (see below).
- For each resource:
  - GET, POST, PUT and DELETE (if applicable) operations on one webpage.
  - Resource URI
  - Parameters:
    - Mandatory and optional in separate tables
    - For each parameter: description + example values
  - Error responses (number + description)
  - Example request and response (linked to API explorer)
  - Links to related resources (e.g. parent or child resources)

Part 3. API explorer

Reading documentation is a last stop rather than a first stop for many developers. Developers prefer to start querying and navigating through the API without having to worry about keys, licenses, tools, etc. Many famous APIs (e.g. Facebook or foursquare) have a so-called API explorer (sometimes called API console) for this purpose. Typically, this API explorer has an intuitive interface through which developers can play around with the API by generating queries, testing results, and uncovering possible hiccups in the process.

In a way, the Waag's web page on http://dev.citysdk.waag.org/map/ is already a simplified API explorer.

Including an API explorer in the CitySDK API documentation may help to convince developers to start trying and using the CitySDK APIs.

- More information and support for building API explorers can be found on http://apigee.com/docs/enterprise/content/whats-api-console
- Examples:
  - https://developer.foursquare.com/docs/explore#req=users/self
  - https://developers.facebook.com/tools/explorer?method=GET&path=1238327819%3Ffields%3Did%2Cname

4 CitySDK Participation pilots preparations

What is the status of the lead API implementation?

Lead API has implemented most of the functionalities that were specified in the CitySDK Smart Participation specification. All functionalities on submitting service request have been implemented. On querying service requests there are features e.g. location based query and updated time-based queries, which have not been implemented yet. These are going to be implemented during second half of 2013.

The lead API was published on 14th of May and documentation can be found from here http://dev.hel.fi/apis/issuereporting. At the moment ten app developers have asked for a test API key.

Is there a (lead) app developed that uses the API?

The lead application Metro's Pitäiskö fiksata (fix my street) –service was developed and launched on 19th of February. It uses the API for submitting and querying status of the individual service requests.

Metro has done also a mobile version of its service. This allows submitting service requests directly from the phone. It is a HTML5 / mobile optimized web page. The mobile version also queries all service requests and shows them on a map.

What are the main experiences and lessons learned during the API implementation?

As they say "The devil is in the details". You find small details which also needs to be specified like in our case allowed media filenames (i.e. *.jpg allowed but *.somethingelse not allowed). Also we

found that the TLS version 1.0 which is only supported in Helsinki endpoint causes problems in some platform. Luckily, these can be bypassed.

What is the status of the replica pilot preparations?

Amsterdam

The Open311 solution is live. Currently, the municipality is doing a soft launch to check if everything is working, especially with the case management system of Amsterdam. There are now two app connected to the Open311 solution (app.youtown.nl and www.verbeterdebuurt.nl). They both use http://open311.dataplatform.nl/opentunnel/open311/v21/* as the entry point for the API. The documentation is almost ready to publish through the official channels.

Barcelona

Working on an initial test pilot where it is possible to send service requests on trash bins.

Manchester

The API is being implemented. In the next weeks a first test app will be developed.

Lamia

Lamia is working on implementing Open 311. Android developers have worked on the app that consumes the Lead Pilot API. The android app has been tested against Lead pilot API.

Lisbon

The API is almost completely implemented. In the next month a first test app will be developed.

Province of Rome

Rome is working on adding tools for user profile data collection and use, in case these data couldn't be available via a public service. User profile is generally speaking something that allows all city services to personalize answers.

In our case, the aim is to grasp from user's curriculum and working experience some more info to suggest jobs or courses available in the City area that are better suited with the user real competences and needs.

User profiling could be useful to create statistics on participation (who is reporting what?) and also to enrich the dialogue among Cities and citizens.

This could in future be extended to give tourism or mobility information more detailed, filtered on the user profile (e.g. accessibility issues etc.).

5 CitySDK Mobility pilots preparations

What is the status of the lead API implementation?

The CitySDK Mobility API endpoint has been running live and stable since March 2013. The API runs on a server, hosted and maintained by Waag Society.

The API has a number of services:

- **Read/write**: The API is read all/write restricted. For reading data no key is needed, for writing accounts and API keys are issued.
- The **Match API** enables you to link objects your own data with existing nodes in

CitySDK Mobility.

- The **API Map viewer** gives a quick map based view on available data, also suitable for non developers.
- **CMS**: Access point for data owners and comprehensive overview of available data layers.
- Documentation is ready on Github July 1st.

Is there a (lead) app developed that uses the API?

Three apps are currently developed/in development by Waag Society that use the API.

1. **Visualization:** An evolving visualisation, which uses the live data in the API to visualize the dynamics in the European cities: http://dev.citysdk.waag.org/visualisation/
2. **Now**: PT departure times in your immediate vicinity: http://dev.citysdk.waag.org/now/
3. **Social Travel App**: the pilot app (concept demonstrated in Istanbul), in development.

In addition, during the Future Everything Hackathon Challenge in March, 20 apps were coded that use (or used) the API. See: http://futureeverything.org/ongoing-projects/innovation-challenge/

What are the main experiences and lessons learned during the API implementation?

- **Available Data**: the available data are limited in scope and granularity. Even when file formats are reasonably standardized (e.g. GTFS) the collected data have among them considerable inconsistencies, especially for predicting purposes (Cf. the paper *Towards a multi modal API*). These are traditional issues with open data in general, but in an API like this they come more to the forefront. Data owners also have yet to commit themselves to a SLA, so that users of their datasets are warned upfront when URL's change, servers go offline.
- **Importing Data**: 'simple' datasets can be added ultrafast. In some cases however snapping and mapping takes a lot of time (and algorithms). This is the case when raw data has been deformed on purpose (to serve the goals of the original user) or when the emerging (European) standard is extremely complex (e.g. DatexII for traffic systems).
- **Size of Data**: When working with live (real time) data, the amount of data grows exponentially, which is an issue in terms of serverload etc. Discussions with data owners are then needed to avoid sending static information with each update.
- **Support & Continuity**: The CitySDK (mobility) API seems to work best when local support and maintenance is available. From a business/continuity perspective the unequivocal support of the local city council is essential. As said above, depending on datasets and especially with valuable live data feeds an ongoing contact has to be established with data owners. Last but not least a close relation to the developer community is needed as well. These three issues mean that a web of offline local relations has to be built and maintained around the API. The strategy then should be to implement CitySDK API's as close to these webs as possible. Currently the endpoint in Amsterdam hosts datasets from and services for Helsinki, Manchester, Lamia and Istanbul. For the above reasons this is not ideal and we should strive for local implementations.

- There are big **differences** in the level of thinking about making Open Data accessible, which is a challenge when promoting CitySDK. A big group has essential gaps in basic understanding of Open Data. In addition, policy makers on strategic levels have difficulty understanding the differences between releasing a static file on a webpage, solutions like CCAN and open API's as CitySDK. In disseminating the project a multilevel strategy is needed where promoting the value of open data as such is the departing point.
- **Open Street Map** as a geographical reference layer works very good. In some more remote areas, the OSM coverage is low (i.c. Lamia). It is however quite feasible to update the needed nodes in OSM as part of the implementation.

What is the status of the replica pilot preparations?

Helsinki

Helsinki will implement the City Navigator App (developed by Code for Europe fellows) as a replication app. In addition they are testing Waag Society's 'Now' webapp.

Istanbul

Main focus is on open data advocacy. To this end, IMM and Waag Society are discussing the possibility of exchanging visions and experiences with open data and mobility on a snior level between the councils.

Another main issue is converting the available data to the GTFS format. Weather data should be on their way.

A site visit to Istanbul to implement the API will take place after July 22nd.

A Hackathon will take place at the 2nd half of September or later to promote the local API.

Lamia

Lamia is researching the possibility of a local instance of the API.

They are also working on issuing PT info in GTFS (with support by Waag Society). Then the automatic updating of the GTFS layer has to be worked out with the local data owner.

Have updated OSM so it is usefull as a layer in CitySDK.

Manchester

Focus in Manchester is on getting a local organisation to commit themselves on hosting the API. A number of site visit have been planned, to give presentations on the features and benefits of CitySDK.

Rome

Outstanding tasks for Rome:

- Creating an application in order to produce the GeoJSON file for describing the whole road map of the Province
- Creating an application in order to produce, every 10 minutes, the GeoJSON file containing real time traffic data (regarding only some main roads)
- Installing the above applications on Province's servers, testing and runtime control
- Publishing the GeoJSON files on the OpenData portal
- Creating an application in order to produce the GeoJSON file for describing the public transport network of the city of Rome

- Publishing the above file on the OpenData portal
- Providing, to the WP4 pilot, all the data of tourism (hotspots, libraries, museums, archaeological sites, etc.) produced in JSON format for the WP5 pilot

6 CitySDK Tourism pilots preparations

What is the status of the lead API implementation?

A reference implementation of a Tourism API server was developed by ISA. This server was deployed for the city of Lisbon and made available for other cities (from the project or outside the project) to use. The server was implemented in C#, making use of ServiceStack and mongoDB. It provides the data access and search functionality provided by the API. In order to use it, cities only have to implement data adapters, which retrieve data from their current systems and publish them into the server. An instance of this server will also be used to run the directory service, which will enable applications to roam among cities and locate the appropriate API endpoint to use at each location.

Lisbon municipality worked on providing access to quality data on Points of Interest and Events. These are retrieved from different systems and have to be related, as events take place in Points of Interest. Data cleaning and quality assurance procedures have begun to be developed and put into place in order to assure high quality data for applications to use. An effort to translate some of the content to English is also in progress. Information on Thematic Routes was not yet available on any system. To this end, Lisbon municipality worked on gathering and defining a set of Thematic Routes for publishing through the tourism API.

A website, available at citysdk.ist.utl.pt was created to publish the work performed. This website includes documentation on the API, documentation on the client libraries, the client libraries, the client applications, the server reference implementation and its documentation. These resource enable either developers to create new apps or cities to publish their data.

Besides the website, other dissemination efforts have been carried out by Alfamicro and CML, which began the preparation of a large developers event to take place in Lisbon.

Is there a (lead) app developed that uses the API?

As the target audience of the Tourism API are developers, IST worked on building tools for use by developers. Client libraries were written for Java, Objective-C, PHP and JavaScript. These libraries facilitate the process of writing applications, lowering even more the barrier of entry to new developers and the development costs, some of the goals of CitySDK. As a proof of concept, some applications were developed, in order to showcase the functionalities provided by the CitySDK Tourism API: an Android application allows Points of Interest, Events and Thematic Routes to be consulted using a map or a list; an Layar layer allows information on Points of Interest to be displayed, overlayed on video, thus providing an augmented reality view of the user point of view; a calendar web widget displays upcoming events; a web widget based on google maps enables geographical searches to be performed on all types of data provided. Currently an iOS application is being developed. For the developers benefit, the web widgets running on the project webpage

provide a debug console, enabling developers to see the queries and replies of the server, thus providing a way to explore the API.

What are the main experiences and lessons learned during the API implementation?

The development of a new API needs to provide access to external staff of the municipality to a development environment, which sometimes is difficult to obtain under the restriction of the municipal IT Departments.

The data sources are sometimes in unexpected repositories inside the municipality, it is worthwhile to double check if the information you are opening is the most valuable for developers and the one that ensures minimum effort from developers to create the API.

What is the status of the replica pilot preparations?

The status in different replication pilots is different from pilot to pilot. In *Helsinki* and *Lamia* implementation will begin soon and according to the specifications of the API reported in D5.1.

In *Amsterdam*, the creation of a mobility API that includes POI is generating some resistance to the implementation of a new API. This issue is still to be clarified together with the project coordination.

In *Rome* political changes in the Regional Government lead to changes in the project team that created a discontinuity in the project work. The WP5 team are confident though that this can be recovered in the near future leading to a successful pilot implementation.