Transactions on Emerging
Telecommunications Technologies

CitySDK Tourism API - Building value around open data

Journal:	*Transactions on Emerging Telecommunications Technologies*
Manuscript ID:	Draft
Wiley - Manuscript type:	Special Issue - Smart Cities
Date Submitted by the Author:	n/a
Complete List of Authors:	Sousa, Pedro; Instituto Superior Técnico, Computer Science and Informatics Lopes Pereira, Ricardo; INESC-ID, ; Instituto Superior Técnico, Computer Science and Informatics Oliveira, André; ISA - Intelligent Sensing Anywhere, Monsieur, Geert; European Research Institute in Service Science,
Keywords:	open data, smart cities, tourism, API, citysdk

TRANSACTIONS ON EMERGING TELECOMMUNICATIONS TECHNOLOGIES
Trans. Emerging Tel. Tech. 0000; **00**:1–11
DOI: 10.1002/ett

Special Issue - Smart Cities

CitySDK Tourism API - Building value around open data

Pedro Cruz Sousa[1], Ricardo Lopes Pereira[2,1*] André Oliveira[3] and Geert Monsieur[4]

[1]Instituto Superior Técnico, Avenida Rovisco Pais 1, 1049-001 Lisboa, Portugal
[2]INESC-ID, Av. Prof. Dr. Cavaco Silva, 2744-016 Porto Salvo, Portugal
[3]ISA - Intelligent Sensing Anywhere S.A, Rua D. Manuel I, 30, 3030-320 Coimbra, Portugal
[4]European Research Institute in Service Science (ERISS), Tilburg University, Warandelaan 2, 5037AB Tilburg, The Netherlands

ABSTRACT

Tourism is a major social and cultural activity with a relevant economic impact. In an effort to promote their attractions with tourists, some cities have adopted the open-data model, publishing data for programmers to use in their own applications. Unfortunately, each city publishes touristic information in its own way.

A common API for accessing this information would enable applications to seamlessly use data from several cities, increasing their potential market while reducing the development costs. This would help developers in making cross-city applications, lowering the overhead of supporting new cities and providing cities with increased exposure. Finally, tourists will also benefit from better and cheaper applications due to the boosted competition.

This article presents the CitySDK Tourism API, which aims to provide access to information about Points of Interest, Events and Itineraries. It was designed in order to be used by municipalities, regional or national governments as well as other public or private entities interested in publishing touristic information. The API comprehends a delegation model which enables one server to guide an application towards other servers which have more detailed information, effectively allowing applications to access worldwide information by only knowing a single API endpoint.

The API was created and validated in the context of the CitySDK project, through which a server reference implementation, client libraries and a set of demonstration applications have also been made available, in order to facilitate adoption by cities and application developers.

*__Correspondence__

E-mail: ricardo.pereira@inesc-id.pt

1. INTRODUCTION

1.1. Motivation

Tourism is a very important social, cultural and economic activity. According to the World Tourism Organization, in 2012 tourism was responsible for 9% of the world's Gross Domestic Product (GDP) and for 1 in every 11 jobs. Tourism generated over 1.3 trillion US$ in exports and accounts for 6% of the world trade. 2012 was the year where the number of international tourists crossed the 1 billion mark*. More than half of international tourists elect Europe as their destination [1].

As vacation time is limited and tourism is a costly activity, tourists wish to make the most of their stay. There is an industry around travel guides, maps and advice. This business is also being explored on the Internet and now is making the crossing to the ubiquitous smartphone. The lower cost of entry

*http://media.unwto.org/sites/all/files/images/unwto1billioninfographic2.jpg accessed June 17^th, 2013

Prepared using ettauth.cls [Version: 2012/06/19 v2.10]

when compared to traditional publishing should foster innovation, harvesting the opportunities brought forward by interactivity, positioning systems, wireless Internet access, augmented reality, social networks and crowd-sourcing. However, often the foundation for tourism applications continues to be accurate, high quality, reliable information from authoritative sources.

National, regional and city authorities compile large amounts of information to use in their internal processes. Municipalities understand the value of these data and many have gone through a multi-step process for sharing these data with tourists in order to improve their experience and attract them to the city. First municipalities created applications (web or mobile) for sharing data with the tourists. This approach is costly and unsuitable for most cities: the number of visitors is not large enough to recuperate the investment; municipalities are not software houses, being unable to keep up with the pace of innovation. Furthermore, municipalities are limited in the types of applications they can provide: e.g. publishing negative opinions written by users about an attraction could expose them to liability.

Making data available to third parties is often a better investment and many entities have followed the open-data path. With access to data, programmers bear the costs and risk, but are free to integrate data from several sources to create novel applications. Market forces should drive innovation, creating the applications tourists want. By publishing data openly, municipalities may also aid in the creation of local businesses exploiting this data [2]. Municipalities may still be forced to publish some applications on their own, should they deem them necessary, but the market not believe them lucrative. Still, the open-data model is not without flaws as each entity publishes different datasets using different data formats. Applications will be restricted to the cities which provide the data required for that application. Furthermore, programmers are forced to invest into dealing with the particularities of each data representation format, thus limiting the number of data sources which can be included into an application. These factors will limit the cities covered by applications and thus their potential market, limiting the number and size of the investments. The local nature of the applications will make them difficult for tourists to find, as they must discover the particular application for each city visited. This problem is also felt for applications created by the municipalities.

The path taken by municipalities has also been traced by other entities related to tourism, such as national and regional governments, museums, concert halls or cultural events organisers.

1.2. CitySDK

If the same touristic data was made available in a single format by several entities, programmers would be able to reach larger audiences with a smaller investment. This would increase competition and tourists would benefit from a wider choice of applications. Awareness about the problem exists and efforts are being made to solve the current open-data fragmentation issue.

Smart City Service Development Kit and its application Pilots (CitySDK) is an European ICT PSP project involving 29 partners from 9 countries. The cities of Amsterdam, Barcelona, Helsinki, Istanbul, Lamia, Lisbon, Manchester and Rome are involved. One of the most important goals of CitySDK is to create an ecosystem in which the work of an application developer is facilitated by having unified and open city interfaces available across different cities in Europe. This means that it should be relatively easy for developers to make use of touristic data coming from multiple European cities, because in such an ecosystem data access is open and unified. CitySDK designs and develops three different unifying APIs: one for participation services (e.g. FixMyStreet), one for mobility data (e.g. public transport data) and one for touristic information.

In this article, we present the third API, referred to as the *CitySDK Tourism API*, and show how it addresses the problems that municipalities and developers face. The CitySDK Tourism API enables access to information about Points of Interest (POIs), events and thematic itineraries. It can be implemented by municipalities (the main focus in the scope of the CitySDK project) other government levels and other private or public organisations such as museums or concert halls. Endpoints for the CitySDK Tourism API are being implemented in Amsterdam, Helsinki, Lamia, Lisbon and Rome.

Trans. Emerging Tel. Tech. 0000; **00**:1–11 © 0000 John Wiley & Sons, Ltd.
DOI: 10.1002/ett
Prepared using ettauth.cls

1.3. Document structure

This document is organised into six sections. In the next section an overview of efforts into providing normalised access to city data is provided. Section 3 details the CitySDK Tourism API. Section 4 describes the reference server implementation and the several client libraries which were produced to make the API easier to use. Opportunities and usage examples of the API are presented in Section 5. Finally, Section 6 presents the conclusions and describes future work.

2. RELATED WORK

The need for common data representation or APIs for tourism data has been present for some time. In this section we present two major efforts in this field: EventsML-G2 and World Wide Web Consortium (W3C) POI Working Group (WG). We also discuss the Open311 GeoReport API as an example of a smart city API which has successfully been implemented across several cities.

2.1. EventsML-G2

EventsML-G2 is a standard for collecting and distributing structured event information, aimed at conveying event information in a news industry environment [3]. This standard is a member of the family of the International Press Telecommunications Council (IPTC) G2-Standards, built on a structural and function framework called the IPTC News Architecture (NAR), and shares many of its components with the other standards of this family. Additionally, the EventsML-G2 makes use of well known industry standards, since its syntax is built on W3C's XML Schema and fully complies with the basic notion of the Semantic Web, the Resource Description Framework (RDF).

One of the main features of the EventsML-G2 standard is it ability to transmit information (i.e. facts) about a specific event. Its comprehensiveness and extensibility makes the standard suitable for covering a big magnitude of event types and cover multiple information facts about a specific event either by literal text (i.e. free text) or by codes from specified vocabularies.

Event though EventsML-G2 can convey information about a POI where an event takes place, that is not its focus. Moreover, as it strives to represent all types of events, EventsML-G2 is a complex standard, making it difficult to implement and use.

2.2. W3C Point of Interest WG

The W3C is an international community where member organizations, a full-time staff, and the public work together to develop Web standards. This community is led by web inventor Tim Berners-Lee and CEO Jeffrey Jaffe.

One of its WGs is the W3C POI and its mission is to develop technical specifications for the representation of POI information on the web [4]. Its Core Recommendation draft defines a generic, flexible, lightweight and extensible POI data model, and one normative syntax for the data model based on Extensible Markup Language (XML). Although XML is the primary model for this specification, other formats are also possible, such as JavaScript Object Notation (JSON).

The data model is shown in Figure 1. It comprises six entities:

- **POIBaseType** is the common entity from which the majority of POI entities are derived. It provides basic properties related with its authorship, licensing, modification dates and identification allowing each element to carry distinct information;
- **POITermType** is an abstract entity derived from POIBaseType and adds properties for the management of categorical descriptions (such as the ones seen in category), link, label, author, license and time properties of POIType;
- **POIType** is an abstract entity derived from POIBaseType and adds entities for describing, labeling, categorizing and indicating the time span of a POI or group of POIs. This entity also includes linking elements to other POIs, external web resources or metadata;
- **Location** is an entity that inherits from POIBaseType and provides a flexible description of the location of a POI. A Location can be represented using geodetic coordinates for the center of the POI, line, polygon, civic address, undetermined (representing unresolved locations) or bounding box (relationship element);
- **POI** inherits from POIType and adds the Location entity for describing the location of the POI;

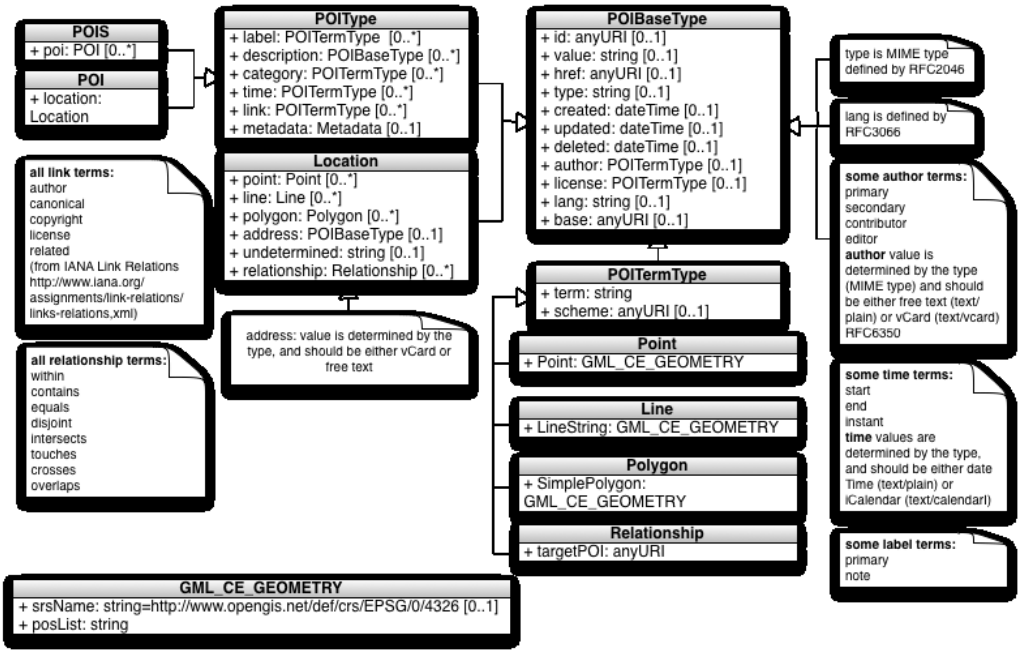

Figure 1. W3C POI Core Data Model [4]

- Finally, **POIS** also derives from POIType and can have one or more children entities of type POI.

This model is flexible and extensible enough to be used within CitySDK Tourism API to model the various types of data (POI, events and thematic itineraries) required. Its use is described in Section 3.1.

2.3. Open311 GeoReport API

The Open311 GeoReport API [5] is a standard that is quite popular among cities in the United States of America. The API allows to both view and report issues which government entities like cities are responsible for addressing. Traditionally, such issues have been handled by custom web forms or phone based call centers (sometimes using the 311 phone number or other short-code). The Open311 GeoReport API is designed to allow both government and third party developers to create new applications for submitting and viewing the reports. The current specification is focused on location-based non-emergency issues such as graffiti, potholes, and street cleaning.

The API is a RESTful API over HTTP that supports both XML and JSON. It consists of two main kinds of resources: *services* (i.e. things that a municipality can offer to a citizen) and *service requests* (i.e. an issue reported by a citizen). The methods that are included in the API support a sequence of interactions where a person first sees a list of available service types, selects one and creates a new service request, and then is able to track the status of that and other requests. However, each method can be accessed independently and some applications may only be focused on querying existing service requests to do analytics. Similarly, the CitySDK Tourism API supports different kind of interactions (e.g. listing routes, listing the POIs of a route, retrieving a POI, etc.).

3. CITYSDK TOURISM API

In this section, we will describe some of the key features of the CitySDK Tourism API. We will describe the message format model, how the API is designed and the features enabled by this design.

In order to address the broader goals of CitySDK (see Section 1.2) a sound methodology is of critical importance to avoid the pitfalls of deploying an uncontrolled maze of APIs [6, 7]. The well-adopted service oriented architecture (SOA) development lifecycle (SDLC) [6] provides a solid foundation for service enablement in an orderly fashion so

DOI: 10.1002/ett

that services can be efficiently used in SOA-based smart city applications. The methodology is based on continuous refinements using a closed loop approach that facilitates designing SOA solutions (e.g. European smart city app) as assemblies of services in which each service assembly is a managed first class aspect of the solution, and, hence, amenable to analysis and change. Developers can then view a smart city app(lication) as a choreographed set of service interactions.

SDLC relies on three fundamental SOA design principles: coupling, cohesion and granularity. These design principles need to guarantee that services are self-contained and come equipped with clearly defined boundaries and interfaces to allow for service composability. Standards and reference models (e.g. W3C POI) are crucial in coping with such design principles. Typically, these standards improve SOA design by defining (sector-specific) business concepts that have a high-degree of cohesion and low-degree of coupling. The design of the tourism API is fundamentally based on W3C POI (see Section 3.1). By reusing existing standards such as W3C POI the chance for adoption is substantially increased. Furthermore, it reduces coupling because developers are not bound to use message formats linked to a specific service implementation [6].

Before diving into a deeper description of the CitySDK API there are three fundamental aspects that should be mentioned. The API provides the following: four types of data models, methods to retrieve information concerning these same data models and also, methods and description fields that retrieve the relationships between each of them. Regarding the data models we provide the following:

- **POI** describes various places in a given city, ranging from monuments and museums to eating places and cultural venues;
- **Event** describes cultural events that happened or are about to happen in the city;
- **Itineraries** describe a group of POIs organized in such a way that they form an itinerary of a given topic (e.g. the life of a given person, the history of a given region or even just specific sightseeing spots);
- **Categories/Tags** describe a list of available categories and tagging terms for each of the aforementioned models.

Each model can also be grouped into a list of its own type, that is, each data model has a listing model where each element is either a POI, Event, Itinerary or a Category/Tag.

We now present the key features of the API.

3.1. W3C POI Model in the API

As mentioned before, our API provides four data models: Points of Interest, Events, Itineraries and Categorization data. These models are mapped using the W3C POI Model presented in section 2.2. We will now present how we modelled each data type.

The **Point of Interest (POI)** is the most easily modelled element of the API. Since the W3C POI Model is specific for this type of data, we used its already specified properties to map our data model. So, the points of interest are mapped and described by using the *POI* entities directly. It should be mentioned that, since the POI is somewhat very detailed and verbose, we defined two granularities for this element: a minimal description, that only includes the key essential properties and a complete model, which is the original W3C POI Data Model. The minimal model is used to map each element of a list of *POIs*. Such list is described by the *POI* entity, but it does not use the descriptive properties of *POIType*.

The **Events** are modelled the same way the POIs were, but instead of having a *Location* entity completly specified, we used the *relationship* property of the same entity to relate a given Event to a POI and omitted the *address* and *undetermined* properties. So, we have an Event completly described using the *POI* entity and use the *relationship* property to also specify and descibe the location of the Event. An Events list is modeled using the *POIS* entity, much like the POIs, but it does not have a different granularity and the root name is *event* instead of *poi*.

The **Itineraries** data model is somewhat more complex. It is defined by using the *POIS* entity and all of its descriptive properties. So, we have the description of the Itinerary itself by using the *POIType* descriptive properties and have the group of POIs by using the *poi* property named as *pois* (so not to confuse with the mentioned list of POIs). It should be mentioned that these POIs are not the original POIs, but are described in the context of the Itinerary, though they include the relationship with its original counterpart, so to fetch the actual description. Finally and like the previous two, the Itineraries has

a list associated with it. Much like the POI, it has a second granularity - a minimal version - in which only the description of each Itinerary is included and their POIs are omitted.

The **Categories/Tags** are equal in nature, but a Category provides a recursive format that the Tags do not. Both borrow from the W3C POI Data Model, but their format is more specific to the needs of the API, rather than following the mentioned model. So, a Category simply follows the *POIType* entity and allows recursiveness and a Tag borrows its properties from the *POIBaseType* to specify a language and value.

At last, most of the terms used in the *POI-TermType* are the suggestions made by the working group itself. However, we've added five more terms regarding price, waiting time, occupation and accessibility information for handicapped people. Such terms are identified as X-citysdk/price, X-citysdk/waiting-time, X-citysdk/occupation and X-citysdk/accessibility-textual and X-citysdk/accessibility-properties, respectively.

The presented models are used in the message format of the API, in the format of JSON.

3.2. API Description

The API follows the Representational State Transfer (REST) architectural design [8]. Thus, we designed a RESTful API over HTTP using JSON. Hence, for each of the presented models we designed various methods to obtain certain data following certain parameters. Many of the parameters are common between the POIs, Events and Itineraries, such as the ability to search for each one of them using a categorical reference, a description or using coordinates. Also, we have provided limitation parameters to allow applications to lazy load the data presented by the API. Of course, there are some parameters that are specific to the data model (e.g. if we search using a description of the POIs, one can ask for either the minimal or complete version or in the case of the Events, we can search using time spans). Furthermore, and for both POIs and Events, one can also search for the relation of a single POI/Event with other POIs/Events. One final method is the ability to search using QR Code or a one-dimensional barcode. Using a single method and providing the textual or bar code information, we retrieve any POIs, Events or Itinerary that match such information.

As for the categorization models, we have provided methods to retrieve the categorical information for each of the aforementioned models by using a list parameter indicating which type of data we wish to retrieve. Such results can also be limited by using the same limitation parameters mentioned above.

Another feature of the API, and somewhat ignored by many REST systems, is complying to the Hypermedia as the Engine of Application State (HATEOAS) constraint of Fielding's dissertation [9]. This constraint states that a client interacts with a network application entirely through hypermedia provided dynamically by the application servers. Therefore, it needs no prior knowledge about how to interact with any particular application or server beyond a generic understanding of hypermedia. We made use of this constraint in three ways:

1. From the entry URL - the only URL that the client needs to know - we present the resources made available by the visited server;
2. Each of the Data Models has an identification (specified by a base URL and ID) that allows to fetch information about that specific model;
3. The POIs, Events and Itineraries can be further described by using a *described-by* (in the links property) which indicates an entry point to another server, which can provide further data on that specific entity.

The mentioned resources indicate to the client, which models and which parameters are available in a given server. These models can be presented by the key-values found in the response message (e.g. if a *find-poi* key is found, then the server supports POIs). As for the allowed searching parameters we have used the URI Template RFC [10]. This way, we allowed a simple yet descriptive manner to indicate which parameters and which data are available and can be fetched.

This feature allowed our system to delegate responsibilities, as we will see next in Section 3.3.

3.3. Delegation

As mentioned in Section 3.2 we designed an API based on hypermedia content. Such feature allows the use of delegation between the various entities involved in the system (e.g. a world-wide directory containing the endpoints of CitySDK-enabled servers); the city servers

providing CitySDK data; and a more granular server provided by museums or any other venues which give more detailed information about that specific POI, Event or Itinerary. Figure 2 shows a diagram of the interactions between each entity.

Further explaining the diagram, a client visits the world-wide directory and provides a given geodetic coordinates and an API key. The world-wide directory then directs the client based on the provided set of coordinates and on the validation of the API key.

The client then visits the provided server address, in which the resources are provided, ending by querying the given server for any of the available data.

At last, the client may further detail the information given by the server by visiting a museum or venue's server. Such server can be known if the provided resource has a *described-by* parameter in the *links* property containing the endpoint URL.

4. IMPLEMENTATION

This section describes the implementation details of the API framework and, in specific, the details of the implementation at Lisbon municipality.

In general terms, the implementation of the platform was planned having in mind key points to ensure its long term sustainability and scalability, adapted not only for the demands expected for the CitySDK project during its development and deployment lifespan but also for its expansion to other cities with different profiles, in terms of expected client demand and provided volume of open-data.

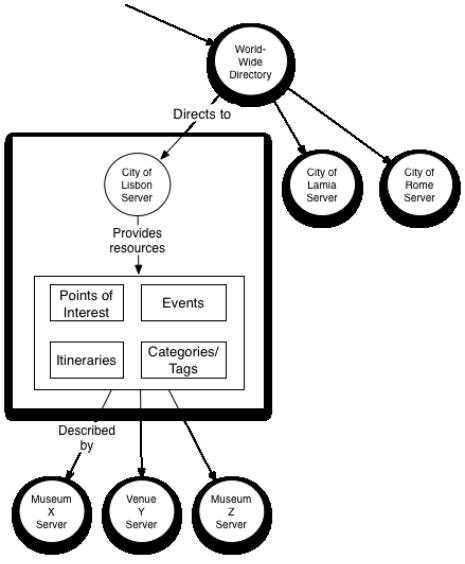

Figure 2. Delegation Model

4.1. Platform Architecture

Figure 3 shows the architectural components of the implemented CitySDK platform.

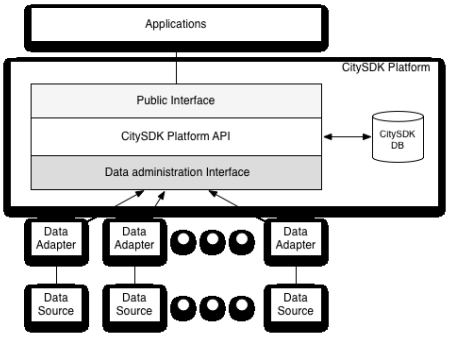

Figure 3. CitySDK platform architecture

In the platform's architecture, the city acts as a data provider which offers one or more relevant touristic information datasets. The CitySDK platform is capable of incorporating several distinct data sources by using a modular approach where each one of the data sources provides its information through a Data Adapter module. The Data Adapter module is a component developed to retrieve raw data from a data source in its native format (XML, JSON, CSV or other) using the data source's native access form (web service, file, database or other) and provide the touristic data (regarding POIs, Events and Itineraries) to be stored in the platform in the data format presented earlier. The Data Adapter module interacts with the CitySDK platform by invoking its Authenticated data administration REST API for manipulating (inserting, updating or deleting) data elements. The Data Adapter module is not only responsible for inserting data elements in the platform but also for maintaining them by updating or deleting the data elements as needed through the platform's life. In most common cases, each data source should have its Data Adapter module but it is also possible to foresee a scenario where a single Data Adapter module retrieves from several data sources, if they are similar in terms of access and provide data in a similar format. Despite not being the case for the implementation of the

platform in Lisbon municipality, nor in the core scope of the CitySDK project's requirements, the platform owner may allow third party data sources (i.e. not controlled by the platform owner such as commercial partners) to insert and maintain data in the platform. The platform is capable of managing several distinct data owners, by using its authentication and data ownership mechanisms.

Even though the CitySDK's platform has a single API endpoint, it can be divided into two distinct conceptual APIs: the public API and the data administration API. The public API is the API with which the client end-user applications (e.g. mobile or web applications) interact to retrieve the touristic data available in the platform, by using the available access and search methods. Although not implemented at the time of writing, an enhancement is planned for this public API by providing an access key system, similar to the one found in some third party interned based services (such as Google Maps), enabling keys to be provided to application developers for use in their applications to access the platform. The intent of this access key system is not to restrict access to the CitySDK platform (as a non-keyed access may be available with limited/lower quality of service) but to prevent abuse and at the same time provide some feedback to the platform owners on the used client applications. The second API, the data administration API, requires a valid authentication to access it, as it is the interface to be used for the purpose of manipulating data elements (POI, Events, Itineraries and their respective categorical information) in the platform. This data administration interface is used by the Data Adapter modules to maintain their data in the platform, enabling the operations of inserting new data elements, updating or deleting existing elements. All the data elements operated through the data administration API are verified against a set of rules and constraints needed to maintain the sanity of the platform and its data.

The CitySDK platform maintains data retrieved in its own database. As the platform must be capable of serving a high number of requests per interval of time, special care was taken when choosing and designing the platform's database. In order to optimize the API and database load to increase the platform's potential throughput, data stored in the database is in a format similar to the W3C POI format (with some minor needed variations), almost ready for being served promptly to the requesting clients, requiring the minimal processing effort as possible in both the platform's database (in terms of number of queries and query complexity) and platform's API engine (in terms of needed processing effort to build the response to the clients).

4.2. Implementation details

As shown in the architecture representation (Figure 3), the core of the CitySDK platform is based on a main component, the application service denoted as "CitySDK Platform API". This provides the previously mentioned APIs (public and data administration), and is aided by a platform's database.

The CitySDK application service was implemented taking in account the CitySDK platform's requirements in terms of providing a REST interface and having high information throughput capability to serve its end-user clients. The application service was implemented using as base a fast and lightweight open-source REST Web Services framework named ServiceStack[†]. The ServiceStack base framework is responsible for handling the protocol part of the CitySDK's platform interface, converting the received web requests into simple method calls and converting back the method return values into a web response. Despite being possible to run the ServiceStack base framework in Mono[‡] making a Linux based environment a possible hosting solution, for the Lisbon's platform instance, a Windows Server 2012 environment in conjunction with the Microsoft Net framework solution was used. This remains the recommended platform while no extensive testing is performed on the platform using the alternative Linux/Mono solution.

For the CitySDK platform's database, an open-source high performance document database named mongoDB[§] was used instead of the traditional relational SQL based database. The choice of this database was mainly related with the high performance for the foreseen database request types. This document database is the CitySDK platform data backend and stores both the platform's data models and some minor administrative data (e.g. access credentials). The data models are stored in an optimal format, taking advantage of the characteristics inherited

[†]http://www.servicestack.net/
[‡]http://www.mono-project.com/Main_Page
[§]http://www.mongodb.org/

DOI: 10.1002/ett

from the fact that the database in use is a document database, aiming to require the minimal effort for the platform's engine to adapt the retrieved data elements to the replies for the clients requests. Although the used database is not a relational database, it also provides the possibility to index any of the stored document's attributes to enable the possibility of performing quicker indexed searches. The database also has the desired capabilities of a Geographical Information System (GIS) enabled database such as PostGIS[¶], enabling the possibility to perform geographical queries to the data such as the ones required to search for a data element within a specified polygon or within a distance from a specified point in the city. The flexibility of the stored data structure is also a valued property of the database, enabling the possibility to have stored objects with different characteristics, as permitted by the W3C POI model used in the CitySDK.

Although Data Adapter modules are not part of the CitySDK platform's itself, they are essential, as they perform the important task of populating it with valuable data. These modules have to be CitySDK-compliant on the data output side and datasource-compliant on the data input side, transforming the datasource's data from its native format into the W3C POI format suitable for insertion into the platform. For the case of the Lisbon implementation, a single database containing aggregated data from POIs and Events (including the relation of the POI where Events occur whenever possible) was identified, so only a Data Adapter module was needed to be implemented for the two types of data. The implemented module runs as a service and, as the volatility of the data is very low (at most a few records are updated each day), the data updating process runs once a day during the empty hours. The data updating process consists of inserting the new data elements found on the city's database into the CitySDK platform, update any change perceived on the city's database on the CitySDK platform and cleaning up the deleted data elements from the CitySDK platform. Regarding Itineraries, Lisbon municipality provides three distinct static elements (no update is made to them) and publishes them manually to be inserted in the CitySDK platform on the form of text files; these text files are imported to the platform using a Data Adapter that runs once as a stand-alone module. In the future, when Lisbon municipality computerizes the Itineraries data, the Data adapter can be converted to a service similar to the one used for importing POI and Events, which automatically imports/updates the data elements.

4.3. Client-side stubs

As a final mention for the implementation details, we present the developed client stubs.

To facilitate the development cycles of potential developers, we provided four libraries which abstract the use of the API and parsing of the received data. These libraries are very similar in usage, as they provide equal naming conventions for methods to perform requests, as well, as to read any of the received data.

The libraries are written in Java, JavaScript/jQuery, PHP and Objective-C and are available online, as is their corresponding documentation and usage examples. They can be found through the project's website [‖].

5. USAGE AND OPPORTUNITIES

5.1. Proof of Concept Applications

To further test our client libraries and API, we developed a group of key applications in various programming languages and frameworks.

The first application made use of the Java library. We developed an Android application which displays POIs, Events and Itineraries following an user's criteria (e.g. using categorical information and/or using coordinates). It displays the retrieved data in both list and map formats (in the case of the Itineraries, it draws the route itself). Its main goal is to use the API to its fullest and demonstrate the various possibilities for applications.

A set of applications making use of the JavaScript/j-Query library was developed. This set of applications is composed by an Event's calendar widget, which makes specific use of Events and time/categories-related searches and a Map displaying all types of data models.

At last, an Augmented Reality application using the Layar framework[**] was also created by developing a PHP web-server, thus using the PHP library. This application

[¶] http://postgis.net/

[‖] http://www.citysdk.eu/developers/

[**] http://www.layar.com/

DOI: 10.1002/ett
Prepared using ettauth.cls

simply makes use of the position, camera and sensors of the user's device to display information about the surroundings and/or the building being displayed in the screen.

Work is being performed on an IOS application to validate the Objective-C library.

5.2. New Opportunities

As mentioned, this API was developed under the CitySDK project which involved 29 partners from 9 countries. Ideally this API could be extended to the rest of Europe, making it a very powerful tool to make tourism related businesses and applications. CitySDK Tourism API provides business opportunities for application developers, cities, service provides and venue owners.

Application developers are the most obvious beneficiaries of the API. Since each city is integrated with the same API and data models, it is much easier and less costly to create new applications and as a result of our delegation model, integrating new cities has much less overhead and complexity. Competition will drive new uses for the data as well as integration with other data sources. Application development may be financed by advertisement on applications, paid applications or sponsoring. Higher value advertisements, geolocated and targeted specifically at tourists, may create opportunities for both developers and advertisement agencies. Tourists will also benefit from the broader availability of the applications and the quality of the data.

Cities, especially early adopters, will benefit from a competitive advantage which will provide some differentiation, by providing tourists with information for their visit and having that information made available through the applications created by developers.

The same can be said about venue owners, which may wish to provide their own CitySDK Tourism API endpoints, where detailed information about the venue (i.e. POI) and events may be provided. Thanks to the delegation model, applications will find these specific endpoints through the city's server. For instance, a museum can provide more detail information:

1. By providing information regarding pricing, the number of people that can be accommodated, queue's length and waiting time or even information on accessibility for persons with disability. This type of highly detailed data or data with a small timespan would be difficult to be managed by a municipality.
2. By providing information about POI in the interior of the museum (e.g. each work of art could be represented as its own POI). An application would thus be able to provide information (e.g. description, multimedia content) about each work of art on display by using a QRCode or RFID to identify it.

Also, our API provides opportunities to create new businesses that either provide more detailed information or provide services to the other players. A company could provide high quality data on a city or area through a paid API, accessible only to paying developers. Other companies could run CitySDK Tourism API endpoint of behalf of others, lessening the investments required to publish data in this format. For instance, a bar might be interested in publishing its live music events, but be unable to run its own server. It could hire a CitySDK Tourism API hosting provider which would make the endpoint available while providing the bar owner with a simple web page for updating the events information.

Hosting and consulting opportunities will also be available, as specialised companies can provide a better value proposition for cities wishing to open their data using this API.

6. CONCLUSIONS

This article presents the CitySDK Tourism API, which is based on the data model defined by the W3C POI WG. This API provides access to information about POIs, Events and Itineraries, enabling municipalities, regional or national governments as well as other public or private entities to publish touristic information for developers to incorporate into their applications. This open-data model is expected to increase the market for tourism applications while lowering the cost of entry. The increased competition should drive down application costs while increasing their quality, benefiting tourists. Data publishers also benefit from increased exposure, increasing the appeal and visibility of their attractions.

Besides the API definition, a reference server implementation and client libraries for the most popular programming languages have been made available in the context of the CitySDK project. This needs to facilitate the adoption of CitySDK Tourism API outside the project.

In the future, we intend to follow the implementation of the CitySDK Tourism API in several cities and refine the API as new challenges arise. In the near future we plan to develop processes for facilitating the management of a worldwide directory for CitySDK Tourism API endpoints, enabling new cities to easily join this service. A single endpoint is crucial for enabling existing applications to take advantage of new CitySDK Tourism API deployments as they become available.

Developer keys are often used to identify developers, providing a way for server owners to throttle API usage in order to deter heavy-hitters and providing an incentive for programmers to produce efficient code. These can also be used to ban misbehaving applications from using an API. A worldwide directory would only make sense if developer keys are portable among different endpoints. Currently we are working on a model for distributed developer keys issuing which does not compromise these goals.

ACKNOWLEDGEMENTS

The CitySDK project is financed by the European Commission under the Information and Communication Technologies Policy Support Programme (ICT PSP) as part of the Competitiveness and Innovation framework Programme (CIP).

This work was partially supported by national funds through FCT Fundação para a Ciência e a Tecnologia, under project PEst-OE/EEI/LA0021/2013

REFERENCES

1. United Nations World Tourism Organization. Tourism Highlights. Booklet 2012. URL `http://mkt.unwto.org/en/publication/unwto-tourism-highlights-2012-edition`, online, accessed 16 Jun 2013.
2. Vilajosana I, Llosa J, Martinez B, Domingo-Prieto M, Angles A, Vilajosana X. Bootstrapping smart cities through a self-sustainable model based on big data flows. *Communications Magazine, IEEE* 2013; **51**(6):–, doi:10.1109/MCOM.2013.6525605.
3. Kelvin Holland. NewsML-G2 Implementation Guide, revision 5.0. IPTC Standards 16 Nov 2012. URL `http://www.iptc.org/site/News_Exchange_Formats/EventsML-G2/Specification/`.
4. Alex Hill, Matt Womer. W3C Points of Interest Core. W3C Editor's Draft 16 Mar 2012. URL `http://www.w3.org/2010/POI/documents/Core/core-20111216.html`, online, accessed 11 Jun 2013].
5. GeoReport v2. Open311 Stable Specification 18 Mar 2011. URL `http://wiki.open311.org/GeoReport_v2`, online, last accessed 20 May 2013.
6. Papazoglou M, Van den Heuvel WJ. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal - The International Journal on Very Large Data Bases* 2007; **16**(3):415.
7. Monsieur G, Snoeck M, Lemahieu W. Managing data dependencies in service compositions. *Journal of Systems and Software* 2012; **85**(11).
8. Fielding RT, Taylor RN. Principled design of the modern web architecture. *ACM Trans. Internet Technol.* May 2002; **2**(2):115–150, doi:10.1145/514183.514185.
9. Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures. PhD Thesis, University of California, Ervine 2000.
10. J Gregorio, R Fielding, M Hadley, M Nottingham, D Orchard. URI Template. IETF RFC 6570 Mar 2012.