SYNCHRONICITY

GA no:	732240
Action full title:	SynchroniCity: Delivering an IoT enabled Digital Single Market for Europe and Beyond
Call/Topic:	Large Scale Pilots
Type of action:	Innovation Action (IA)
Starting date of action:	01.01.2017
Project duration:	36 months
Project end date:	31.12.2019
Deliverable number:	D4.2
Deliverable title:	Technical Validation (Phase 1)
Document version:	1.0
WP number:	WP4
Lead beneficiary:	9-UC
Main author(s):	Ignacio Elicegui, Laura Rodriguez, Luis Díez, Luis Muñoz (UC), Arturo Medela (TST), Andrea Gaglione, Angelo Capossele, Daniel Puschmann (FCC)
Internal reviewers:	Geoffrey Stevens (FCC), Nuria de Lama (ATOS), Martin Brynskov (AU)
Type of deliverable:	Report
Dissemination level:	PU
Delivery date from Annex 1:	M19
Actual delivery date:	M20 (30.08.2018)

Co-funded by the European Union

This deliverable is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732240.

Executive Summary

According to the Description of Action, WP4 copes with the validation in terms of architecture, services, user acceptance, market and replication. Whilst D4.1 defines the whole methodology to be applied, this document presents the results of the functional and technical validation applied to the data models, interfaces, components and enablers which are part of the SynchroniCity framework. In particular, the mandatory assets which have to be exposed by the reference zones wishing to provide support to the experimenters of the open call are:

- *Common data models*
- *NGSI interface for managing contextual information*
- *Security framework relying on the OAUTH 2.0 protocol*
- *Data marketplace*

Upon the architecture design and specification activities carried out in WP2, it is mandatory to guarantee the adoption and interoperability of the above components aiming at easing replication and integration activities. Hence, several tools have been developed for automatizing the test campaigns which are done at reference zone level and per component.

The results derived from applying the corresponding methodology and tools shape the core of this document. As expected, there are different maturity levels in terms of SynchroniCity framework adoption which in any case represents a major issue. However, this situation means that the potential solution offer will be much richer than in the case of having a monolithic one.

It is worth highlight that this document sets up appropriate links with WP1 and WP6. With the former because it complements, in a quantitative approach, the work that is being performed in terms of monitoring and synchronization of the reference zones (Task 1.3). The links with WP6 emanate from the identification of eventual new key performance indicators as well as the measurement of some of those already jointly agreed.

Last but not least, it has to be noted that a second document, D4.3, to be delivered just before starting the execution of the experiments (February 2019) will complement present results. Besides including further architecture component validation, such document will collect the results related to service validation. The service framework is being designed and developed in WP3 and is expected that the already agreed baseline services will be validated when delivering D4.3.

Abbreviations

AJV	Another JSON Schema Validator
API	Application Programming Interface
BLE	Bluetooth Low Energy
BT	British Telecom
CIM	Context Information Management
CKAN	Comprehensive Knowledge Archive Network
CLI	Command-Line Interface
D	Deliverable
DigiCat	Digital Catapult
EC	European Commission
ESN	Environmental Sensor Network
ETSI	European Telecommunications Standard Institute
HTTP	Hypertext Transfer Protocol
ID	Identifier
IdM	Identity Manager
IETF	Internet Engineering Task Force
IoE	Internet of Everything
IoT	Internet of Things
ISG	Industry Specification Groups
JSON	JavaScript Object Notation
JWT	JSON Web Token
KPI	Key Performance Indicator
M	Month
MIM	Minimal Interoperability Modules
MQTT	Message Queue Telemetry Transport
NFC	Near Field Communication
NGSI	Next Generation Service Interface
NGSI-LD	NGSI Linking Data
OASC	Open & Agile Smart Cities
OAUTH2	Open Authorization 2
ODMS	Open Data Management Systems

OMA	Open Mobile Alliance
PDP	Policy Decision Point
PEP	Policy Enforcement Point
POI	Point of Interest
REST	Representational State Transfer
RFC	Request for Comments
RZ	Reference Zone
SAN	Santander reference Zone
SC_DMV	SynchroniCity Data Models Validator
TM Forum	TeleManagement Forum
TTL	Time To Live
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WP	Work Package
WT	Work Task

Contents

List of Figures

List of Tables

1 Introduction

Work Package (WP) 4 is coping with the validation of the SynchroniCity project in its different dimensions. As such, Task 4.1 [1] has provided the methodology to be followed during project execution for assessing functional and technical issues related to the enablers, components and APIs which are paramount in the adopted framework. But going far beyond this objective, it has also proposed the methodology for validating other characteristics such as replication and integration which will be carried out as part of Task 4.4 upon the pilots developed and deployed during the execution of the open call. Last but not least, it has paved the way for assessing both user perception and market penetration which is now being expanded as part of Task 4.3.

It is up to Task 4.2 to cope with the functional and technical validation of the Minimal Interoperability Modules (MIM) agreed along WP2 progress. Essentially, those MIM embrace the following assets:

- Data models
- NGSI interface for contextual information
- Security framework
- Data marketplace

The validation of the above components is carried out according to the functionalities which are expected to be fulfilled by them. Hence, for validating the data models agreed in Task 2.2 [2] a script has been developed aiming at speeding up the process. It is important to note that the validation has been carried out per reference zone (RZ) and per data model as it will be explained later on. In the case of the interfaces the different primitives have been tested. This applies to the NGSI interface as well as to the security framework, in which the OAUTH 2.0 and its primitives have been assessed according to the corresponding flows and functionalities. Finally, an initial validation approach has been specified for the data marketplace which initially has allowed to validate its instance in Santander RZ.

As part of this WP4 and aiming at setting tight links with WPs 1 and 6, some initial key performance indicators (KPI) have been derived according to the criteria jointly agreed with WP6. Indeed, at this project stage, the number of new/data models proposed/adopted by the RZs is one of the initial KPIs which has been identified.

Two final comments in this section. First, this is the initial validation document which introduces the different RZ status and the extra work needed to support the use cases. As such a second document, *SynchroniCity D4.3 Technical Validation Phase II*, will be submitted just before the beginning of the open call projects' implementation. Such document will also include the validation of the eventual microservices to be developed as part of WP3 which will interact with SynchroniCity framework through the above interfaces.

2 Synchronicity Framework and validation approach

The model developed for the SynchroniCity Framework consists of a set of canonical components and interfaces that are deployed across different cities to let them become part of IoT Smart City Digital Single Market. Through the envisaged model, urban data belonging to different cities can be linked to both develop and validate novel services and applications under real-world conditions.

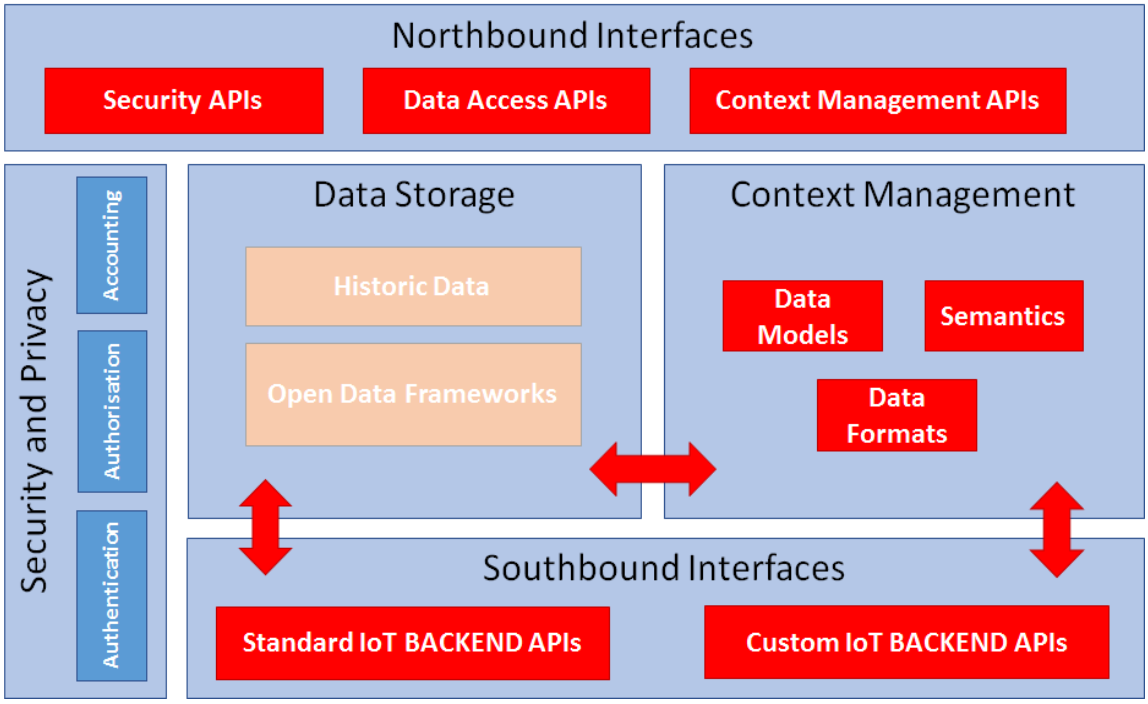

Figure 1. SynchroniCity Architecture's Interoperability Points & Interfaces

The facility has been designed to accommodate state-of-the-art initiatives and to be employed by users of different origins and technical backgrounds (e.g., developers, entrepreneurs, activists). It supports urban service providers' operations, as well as the co-creation and validation of a large set of smart city services and applications. SynchroniCity instantiates at its core the interoperability mechanisms and principles envisaged in the global Open & Agile Smart Cities (OASC) initiative. The tenet is that modern cities and communities need a capacity for systematic experimentation to harness the power of new technologies. This isn't just about technical capabilities, and learning from practice, but also to a large degree a question of opening the innovation and experimentation processes up to the community, in order to truly co-create future services.

The SynchroniCity project objective, through the implementation of this model, is to facilitate the provision of new urban services in the linked cities, that will permit them to seed, sustain and grow powerful ecosystems within the cities facilitating service co-creation and experimentation in line with local needs, leveraging global dynamics. The figure [Figure 1] shows the high-level architecture of the SynchroniCity solution. It is based on a 5-section model, comprising the Northbound and Southbound Interfaces, and the Data Storage, Context Management and Security and Privacy sections.

- **Northbound Interfaces**: grouping a set of APIs that provide the platform functionalities that can be used by the final end users' applications.
- **Southbound Interfaces**: including a set of useful APIs to connect the overall platform to heterogeneous data sources and IoT devices.
- **Data Storage**: in charge of storing data retrieved by the interaction with heterogeneous sources, not only IoT devices and sensors but also Open Data catalogues, as well as maintaining an archive of the historical data generated by those sources.

- **Context Management**: responsible of managing the context information received from IoT devices compliant with the data models and data formats accepted by the SynchroniCity Architecture.
- **Security and Privacy**: integrating triple-A features (Authentication, Authorisation, Accounting) to comply with all the required security aspects.

This information can be extended by consulting the deliverable D2.1.1 [3].

2.1 Interoperability points

2.1.1 Data models

According D2.2 [2], SynchroniCity Architecture will use OMA NGSI [4] Metamodel to base all its information models, included in SynchroniCity pilots and shared through SynchroniCity framework. For the implementation, a JSON Representation was chosen and FIWARE Data models [5] where used as its first approach. From the current and envisioned data sources on each RZ, Task 2.2 has defined on a set of JSON-Schemas to define and describe the final implementation of the SynchroniCity information and data models [6]. Table below [Table 1] shows the datasets and JSON Schemas used for Synchronicity's Data model's validation phase I process.

Entity Type	Recommended Data Model (JSON Schema). Described in [6]	Description
Bike Hire Docking Stations	BikeHireDockingStation	a bike hire docking station where subscribed users can hire and return a bike. It provides data about its main features and availability of bikes and free slots.
Air Quality Observed	AirQualityObserved	represents an observation of air quality conditions at a certain place and time
Parking (OffStreet)	OffStreetParking	an offstreet parking site with explicit entries and exits
Parking (OnStreet)	OnStreetParking	an on street, free entry (but might be metered) parking zone which contains at least one or more adjacent parking spots
Parking Spot	ParkingSpot	an individual, usually monitored, parking spot/parking lot
Noise Level Observed	NoiseLevelObserved	represents an observation of those parameters that estimate noise pressure levels at a certain place and time
Weather Observed	WeatherObserved	represents an observation of weather conditions at a certain place and time
Traffic Flow Observed	TrafficFlowObserved	description of a set of attributes related to traffic flow in a city (identifier, type, location, dateModified, dateObserved, intensity and occupancy)
Bus Line	SAN Bus Datamodels	a bus line in which there are bus stops (mandatory) and buses (optional)
Bus Stop	SAN Bus Datamodels	a bus stop which belongs to a bus line and in which buses can run
Bus Timetables Real Time (estimations)	SAN Bus Datamodels	estimation of arrival times of buses in relation to a combination of busStop-busLine
Point of Interest	PointOfInterest	harmonised geographic description of a Point of Interest
Alert	Alert	an alert is generated by a specific situation. The main features of an alert are that it is not predictable, and it is not a recurrent data.

City Event	Event	events happening in the city (concerts, games, etc.)
Public Transport Fleet Vehicles	Vehicle	vehicles that are used for public transportation (buses, trains, etc.)
Civic Issue Tracking	Open311:ServiceRequest	issues reported by citizens
Bicycle Tracking	TrafficFlowObserved	description of a set of attributes related to traffic flow in a city. Here used to map bicycle traffic.
Waste Containers Assets	Waste Container Model	static properties of a waste container (big belly)
Parks & Gardens sensors	GreenSpaceRecord	harmonised description of the conditions recorded on a particular area or point inside a greenspace (flower bed, garden, etc.)

Table 1. First set of approved SynchroniCity Data models (July 10th, 2018)

SynchroniCity GitLab provides a folder [6] to define new SynchroniCity data models, linked to FIWARE Data models. With the support of both, SynchroniCity and FIWARE community, this folder includes all the details, schemas and attributes of each SynchroniCity approved data model.

2.1.2 NGSI interface for accessing/retrieving contextual information

This interoperability point represents the main way to access Context Management functionalities. SynchroniCity Architecture framework [**¡Error! No se encuentra el origen de la referencia.**] agrees on defining NGSI as the proper interface to interact with Context, relying on standard OMA [4] and incoming ETSI NGSI-LD [7]. The overall approach of this NGSI interface is given by OMA, and divides the offering functionalities in three main sets: Resource Directory (OMA NGSI9) methods, to create, describe and manage Context Entities; Context Information (OMA NGSI10), that query/retrieve context data provided by the context entities; and Context Broker features, (OMA NGSI9 & NGSI10) that supports asynchronous context information notifications by providing Publish/Subscribe mechanisms:

- **Resource Directory**: its purpose is to create the context entities to register and exchange information about the availability of context information. The main interaction types are:
  - Registration (registerContext), description and modification of context entities, i.e. announcements that certain context information is available
  - One-time queries for searching/discovery (discoverContextAvailability) entities where certain context information is available
- **Context Information access**: to exchange context information related to the registered entities. Keeping this simple, its two main interaction types are:
  - search and retrieve context information (queryContext) information provided by a specified context entity
  - update (updateContext) the context information captured by context entities
- **Context Broker**: to implement Context Information publish/subscribe functionalities
  - Context information subscriptions (subscribeContext) for context information (attribute values) updates (and the corresponding notifications)
  - Context Entities Subscriptions (subscribeContextAvailability) for context availability (new entities/attributes/info sources) information updates (and the corresponding notifications)

SynchroniCity Reference framework [**¡Error! No se encuentra el origen de la referencia.**] suggests current NGSIv2 [7] API implementation to support this Interoperability point set of primitives:

2.1.2.1 Resource Directory Primitives

NGSI Operation (OMA definition [4])					
Name	**registerContext**				
Description	allows registering and updating of registered Context Entities				
Primitives	**registerContextRequest**				
	Part name	**Part type**	**Optional**	**Description**	
	ContextRegistrationList	ContextRegistration [1..unbounded]	No	List of ContextRegistration structures. Specifies the set of attributes and associated metadata.	
	Duration	xsd:duration	Yes	Desired availability period.	
	RegistrationId	xsd:string	Yes	Registration identifier used to update previous registrations.	
	registerContextResponse				
	Part name	**Part type**	**Optional**	**Description**	
	Duration	xsd:duration	Yes	Confirmed availability period	
	RegistrationId	xsd:string	No	Registration identifier that could be used to update this registration	
	ErrorCode	StatusCode	Yes	Error reported by the operation	
Reference Implementation (NGSIv2 specifications [9])					
Name	**Create Entity**	Method	POST	URL	/v2/entities
Body	JSON object including *ContextRegistration* structures (set of attributes and associated metadata)				
Response	HTTP Status code	201 (Created) / Others (Error Code)			
	Location	URL of the registered entity (to be updated/modified)			

NGSI Operation (OMA definition [4])					
Name	**discoverContextAvailability**				
Description	allows the synchronous discovery or request of the Context Entities and related Context Information that can be provided				
Primitives	**discoverContextAvailabilityRequet**				
	Part name	**Part type**	**Optional**	**Description**	
	EntityId	EntityId [1..unbounded]	No	List of Modifier to identify the Context Entity(ies) to discover or Id to request for.	
	AttributeList	xsd:string [0..unbounded]	Yes	List of attributes or group of attributes to discover.	
	Restriction	Restriction	Yes	Restriction on the attributes and meta-data of the Context Information	
	discoverContextAvailabilityResponse				
	Part name	**Part type**	**Optional**	**Description**	
	ContextRegistrationResponseList	ContextRegistrationRes ponse [0..unbounded]	Yes	List of Context Registration responses, including all entities discovered matching the ID modifier and the attributes	
	ErrorCode	StatusCode	Yes	Error codes for general operation errors	
Reference Implementation (NGSIv2 specifications [9])					
Name	**Query Entity**	Method	GET	URL	/v2/entities/{id}

Parameters	• type: discovers by type of entities • idPattern: discovers by entities ID patterns • q: to discover by attributes info	
Response	HTTP Status code	200 (Success) / Others (Error Code)
	JSON Object	with the entity/ies and attribute/s matching the modifiers requested

2.1.2.2 Context Information management primitives

NGSI Operation (OMA definition [4])				
Name	**updateContext**			
Description	allows updating/register a set of Context Information, related attributes and metadata: • append: add the new set of attributes to an existing entity context or to a new one. • update: replace the old attributes values for the new values included in the call. • delete: remove the pointed attributes from a specified context entity. If no attributes are pointed, it will remove all context information (data) related to the given entity.			
Primitives	**updateContextRequest**			
	Part name	**Part type**	**Optional**	**Description**
	ContextElementList	ContextElement [1…unbounded]	No	List of Context Elements containing only the subset of Context Information (related attributes (or context domain) and metadata) to be modified.
	UpdateAction	UpdateActionType	No	Indicates the type of action that is performed within the update operation: update/append/delete
	updateContextResponse			
	Part name	**Part type**	**Optional**	**Description**
	ErrorCode	StatusCode	Yes	Error codes
	ContextResponseList	ContextElementRespon se [0…unbounded]	Yes	List of response containing the indication of the Context Element and the related statusCode.

Reference Implementation (NGSIv2 specifications [9])						
Name	**Update Entity**	Method	POST/PATCH/DELETE	URL	/v2/entities/{id}/attrs/{attrName}	
Body	JSON object (POST/PATCH) including context info (attributes and associated metadata) to POST (append) PATCH (update) or DELETE (remove).					
Response	HTTP Status code	204 (Created) / Others (Error Code)				

NGSI Operation (OMA definition [4])				
Name	**queryContext**			
Description	allows the synchronous retrieval of Context Information related to an entity or list of entity identifiers			
Primitives	**queryContextRequest**			
	Part name	**Part type**	**Optional**	**Description**
	EntityIdList	EntityId [1…unbounded]	No	List of identifiers of the Context Entity(ies) for which the Context Information is requested. Identifiers can contain patterns represented as regular expressions.
	AttributeList	xsd:string [0…unbounded]	Yes	List of ContextAttributes and/or AttributeDomains that are queried (Note: If this parameter is absent, the receiver of the request SHALL return all attributes available). NGSI component SHALL return an error if this parameter is required by a service provider policy and is not specified in the message.

	Restriction	Restriction	Yes	Restriction on the result set of the query. Restrictions are based on the values of attributes and metadata of the Context Information

queryContextResponse			
Part name	**Part type**	**Optional**	**Description**
ContextResponseList	ContextElementResponse [0…unbounded]	Yes	List of Context Information, related attributes (or group of attributes) and metadata.
ErrorCode	StatusCode	Yes	Error codes for general operation errors

Reference Implementation (NGSIv2 specifications [9])					
Name	**Query Entity**	Method	GET	URL	/v2/entities/{entityID}/attrs/{attrsName}
Params	Options: KeyValues to get a more compact and brief representation (list of attributes, values...)				
Response	HTTP Status code	200 (Success) / Others (Error Code)			
	JSON Object	List of Context Information, related attributes (or group of attributes) and metadata.			

2.1.2.3 Publish/Subscribe (Context Broker) primitives

NGSI Operation (OMA definition [4])	
Name	**subscribeContextAvailability**
Description	allows the asynchronous discovery of the potential set Context Entities, types of Context Entities and related Context Information that can be provided

Primitives

subscribeContextAvailabilityRequest			
Part name	**Part type**	**Optional**	**Description**
EntityId	EntityId [1..unbounded]	No	List of identifiers or name patterns of the Context Entity(ies) to discover
AttributeList	xsd:string [0…unbounded]	Yes	List of attributes or group of attributes to be discovered
Reference	xsd:anyURI	No	The interface reference for the notifyContextAvailability operation.
Duration	xsd:duration	Yes	Requested duration of the subscription. Negative values SHALL result in an error. In case of the value 0, the context component SHALL only notify the current value and SHALL NOT send subsequent notifications (one-time subscription). If the Context Management component has a policy to always require duration, the operation SHALL return an error in case the parameter is not present. If the parameter is omitted, the Context Management component MAY select a duration and return this in the response.
Restriction	Restriction	Yes	Restriction on the attributes and metadata of the Context Information
SubscriptionId	xsd:string	Yes	Used in the notification message and subsequent requests

subscribeContextAvailabilityResponse			
Part name	**Part type**	**Optional**	**Description**
SubscriptionId	xsd:string	No	The identifier of the subscription
Duration	xsd:duration	Yes	Negotiated duration of the subscription
ErrorCode	StatusCode	Yes	Error reported by the operation

Reference Implementation (NGSIv2 specifications [9])					
Name	**Subscriptions**	Method	POST	URL	/v2/subscriptions

Body	JSON object including: • Conditions as the set of entities/attributes that triggers the notifications (Context Information sources wanted to be subscribed to) • Expiration date (duration) • URL where to send notifications	
Response	HTTP Status code	201 (Created) / Others (Error Code)
	Location	subscription ID used for updating and cancelling the subscription

NGSI Operation (OMA definition [4])				
Name	**subscribeContext**			
Description	will provide subscription (asynchronous notifications) to Context Information, retrieving attributes and associated current values			
Primitives	**subscribeContextRequest**			
	Part name	**Part type**	**Optional**	**Description**
	EntityIdList	EntityId [1…unbounded]	No	List of identifiers of the Context Entity(ies) for which the Context Information is requested. Identifier can contain patterns represented as regular expressions.
	AttributeList	xsd:string [0..unbounded]	Yes	List of ContextAttributes and/or AttributeDomains to which the requestor wants to subscribe.
	Reference	xsd:anyURI	No	URI that identifies the interface where the notifyContext operation SHALL be invoked.
	Duration	xsd:duration	Yes	Requested duration of the subscription. Negative values SHALL result in an error. If the Context Management component has a policy to always require duration, the operation SHALL return an error in case the parameter is not present. If the parameter is omitted, the Context Management component MAY select a duration and return this in the response.
	Restriction	Restriction	Yes	Restriction on the attributes and meta-data of the Context Information
	NotifyConditions	NotifyCondition [0…unbounded]	Yes	Conditions when to send the notifications.
	Throttling	xsd:duration	Yes	Proposed minimum interval between notifications.
	subscribeContextAvailabilityResponse			
	Part name	**Part type**	**Optional**	**Description**
	SubscribeResponse	SubscribeResponse	Yes	Response to the SubscribeContextRequest
	SubscribeError	SubscribeError	Yes	The error reported by the receiver of the request

Reference Implementation (NGSIv2 specifications [9])						
Name	**Subscriptions**	Method	POST	URL	/v2/subscriptions	
Body	JSON object including: • Conditions as the set of entities/attributes that triggers the notifications (Context Information sources wanted to be subscribed to) • Expiration date (duration) and throttling • Set of entities and attributes (values) to be included on the notifications (Context Information) • URL where to send notifications					
Response	HTTP Status code	201 (Created) / Others (Error Code)				
	Location	subscription ID used for updating and cancelling the subscription				

2.1.3 Historical data API

SynchroniCity architecture includes the Data Storage Interface among its interoperability points. According the D2.10 [**¡Error! No se encuentra el origen de la referencia.**], this will cover two defined APIs: the Historical Data API and the Open Data API. This last one is linked to the different Open Data Management Systems (ODMS) portals belonging to the RZs, and retrieves a list of publicly available datasets and catalogues, always according to each portal data models and formats. Initially, as there isn't a standard way to homogenize this information, these sources are kept as additional features and are not considered as part of the validation process. On the other side, the SynchroniCity Historical API provides methods to access historical raw data and aggregated time series related to an attribute of an entity. This API definition (detailed on the SynchroniCity API [10]) is linked to the Context Information API (NGSI Interface) as it follows the same approach and nomenclature to identify entities and related attributes to retrieve. This way, in the SynchroniCity reference framework implementation, SynchroniCity Historical API complements the Context Information management (NGSIv2 API) according the ETSI ISG CIM NGSI-LD draft specification [7].

This API implements a simple GET query to retrieve the requested time series that includes an entity ID and an attribute:

Historical Data Retrieval (SynchroniCity/ETSI ISG CIM NGSI-LD draft specification [7])			
Name	**Get Historical Data**		
Description	retrieves historical data (Time series) related to an attribute of an entity		
Primitive	**Get Historical Data Request**		
	Part name	**Optional**	**Description**
	EntityID	No	The ID of the entity to be requested, according the NGSI data model and as it is registered in the Context Management component.
	AttributeName	No	Name of the Attribute to be requested, as it has been defined in the corresponding data model.
	type	yes	specify the entity type in the cases there is ambiguity using just the ID.
	timerel	No (for temporal queries)	this attribute discriminates a temporal query from a normal context query • *before* to define a temporal query to match the values of the attribute before the time specified in time key parameter. • *after* to define a temporal query to match the values of the attribute after the time specified in time key parameter. • *between* to define a temporal query to match the values of the attribute after the time specified in time key parameter and before the time specified in end time key parameter.
	time	No (if a valid timerel is present)	the starting date and time from which the attribute context information is desired.
	endtime	Yes	the final date and time until which the raw context information is desired. If no endtime is provided current time is the default value.
	timeproperty	Yes	the name of the Property that contains the temporal data that will be used to resolve the temporal query (e.g. timeproperty=dateModified)
	lastn	Yes	only the requested last entries will be returned. It is a mandatory parameter if no limit and offset are provided
	limit	Yes	In case of pagination, the number of entries per page. It is a mandatory parameter if no lastn is provided
	offset	Yes	In case of pagination, the offset to apply to the requested search of raw context information. It is a mandatory parameter if no lastn is provided.
	Get Historical Data Response		
	Part name	**Optional**	**Description**

	Get Response	No	Retrieved temporal series information (If error code = 200). Error description if error code differs from 200.
	Get Error	No	The error code reported by the receiver of the request

Reference Implementation (SynchroniCity specifications [10])					
Name	**Get Historical Data**	Method	GET	URL	/v2/entities/entityId/attrs/attrName?type=&timerel=&time=&endtime=&timeproperty=&lastn=&limit=&offset=
Params	● type (String) ● timerel (Enum: before; after; between) ● time (String) ● endtime (String) ● timeproperty (String) ● lastn (Number) ● limit (Number) ● offset (Number)				
Response	HTTP Status code	200 (Success) / Others (ETSI uri of the error plus description)			

	JSON Object (if 200 - Success is returned)	Id and type of the entity plus the requested list of modifications of the requested attribute including its corresponding date according the schema:
		{
		"id": "",
		"type": "",
		"attribute": [
		{
		"type": "",
		"value": "",
		"modifiedAt": ""
		}
		]
		}			

2.1.4 Security API

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. OAuth 2.0 is the industry-standard protocol for authorization, being developed within the IETF OAuth Working Group [11].

SynchroniCity Security API [12] proposes an OAuth 2.0 interface to grant access mainly to context and historical information, either to read it or to provide it. This way, the OAuth 2.0 API in conjunction with the Identity Manager component (provided by each Reference Zone), will identify and authenticate data consumers and data providers, allowing also the roles' management and restriction accesses.

The different requests will require an Authorization Basic header. This is built with the Client ID and Client Secret credentials provided by the IdM component and referred to the registered application. So the string will be *base64(client_id:client_secret)*. The *redirect_uri* parameter must match the *Callback URL* attribute provided in the application registration.

The OAuth 2.0 grant types are:

2.1.4.1 Authorization Code Grant

The Authorization Code grant method is used by confidential and public clients to exchange an authorization code for an access token. After the user returns to the client via the redirect URL, the application will get the authorization code from the URL and use it to request an access token. It's the preferred mechanism, for instance, to sign into a web app using a valid Facebook or Google account. The authorization code is obtained by using the SynchroniCity's IdM (or the corresponding RZ's authentication server) as an intermediary between the client (the registered application) and

resource owner (the user). Instead of requesting authorization directly from the resource owner, the client directs the resource owner to an authorization server (via its user-agent as defined in RFC2616), which in turn directs the resource owner back to the client with the authorization code.

Step One: Obtaining the Auth. Code:

Authorization Code Request (OAuth 2.0 specification [11])			
Name	**Authorization Code Request**		
Description	Requests an authorization code for an access token.		
Primitive	**Authorization Request**		
	Part name	**Optional**	**Description**
	response_type	No	must be set to “code”
	state	yes	Internal use.
	redirect_uri	yes	if the redirect URI was included in the initial authorization request, the service must require it in the token request as well. The redirect URI in the token request must be an exact match of the redirect URI that was used when generating the authorization code. The service must reject the request otherwise
	client_id	Yes (required if no other client authentication is present)	Identifies the app the access is requested for. If the client is authenticating via HTTP Basic Auth or some other method, then this parameter is not required. Otherwise, this parameter is required.
	Authorization Code Response		
	Part name	**Optional**	**Description**
	Response/Error Code	No	302 Found (If success)
	code	No (if Success)	authorization code
	Callback url	No (if 302 Response)	Callback url
	State	No (if 302 Response)	state (Internal use)
	Error Description	No	HTTP/1.1 error description

Reference Implementation (OAuth 2.0 specifications [12])					
Name	**Authorization Code Request**	Method	GET	URL	/oauth2/authorize?response_type=&client_id=&redirect_uri=&state=
Params	• response_type (string) = code • redirect_uri (string) [callback_url] • client_id (string)				
Response	HTTP Status code	302 (Success) / Others (HTTP/1.1 error plus description)			
	Location	callback_url + code + state			

Step two: Obtaining the Auth. Token

Access Token Request (OAuth 2.0 specification [11])			
Name	**Access Token Request**		
Description	Requests an access token based on a previous retrieved authorisation code.		
Primitive	**Access Token Request**		
	Part name	**Optional**	**Description**
	Client ID	No	with the the client’s ID (provided by IdM)
	Client Secret	No	with the client’s secret (provided by IdM)
	grant_type	No	authorization_code

	code	No	code obtained with Authorization Request
	redirect_uri	yes	if the redirect URI was included in the initial authorization request, the service must require it in the token request as well. The redirect URI in the token request must be an exact match of the redirect URI that was used when generating the authorization code. The service must reject the request otherwise

Access Token Response		
Part name	**Optional**	**Description**
Response/Error Code	No	200 Ok (If success)
access token	No (if Success)	a JWT signed with the authorization server's private key
token type	No (if Success)	string = "bearer"
refresh token	No (if 302 Response)	an encrypted payload that can be used to refresh the access token when it expires.
expiration time	No	with an integer representing the TTL of the access token
Error Description	No	HTTP/1.1 error description

Reference Implementation (OAuth 2.0 specifications [12])					
Name	**Access Token Request**	Method	POST	URL	/oauth2/token
Headers	● Authorization: Basic base64(client_id:client_secret) ● Content-Type: application/x-www-form-urlencoded				
Body	grant_type=authorization_code&code=[auth_code] &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcallback_url				
Response	HTTP Status code	200 (Ok) / Others (HTTP/1.1 error plus description)			
	JSON Object	{ "access_token":["access_token"], "token_type":"bearer", "expires_in":[TTL in seconds], "refresh_token":["refresh_token"] }			

2.1.4.2 Implicit Grant

The implicit grant is similar to the authorization code grant with two distinct differences: it is intended to be used for user-agent-based clients (e.g. single page web apps) that can't keep a client secret because all of the application code and storage is easily accessible; secondly instead of the authorization server returning an authorization code which is exchanged for an access token, the authorization server returns an access token.

Implicit Authorization Request (OAuth 2.0 specification [11])			
Name	**Implicit Auth. Request**		
Description	Simplified authorization code flow optimized for clients implemented in a browser		
Primitive	**Implicit Authorization Request**		
	Part name	**Optional**	**Description**
	response_type	No	must be set to "token"
	state	yes	Internal use.
	redirect_uri	No	must match the Callback URL attribute provided to the IdM within the application registration
	client_id	Yes (required if no other client authentication is present)	Identifies the app the access is requested for.
	Implicit Authorization Response		
	Part name	**Optional**	**Description**

	Response/Error Code	No	302 Found (If success)
	token	No (if Success)	access token
	Callback url	No (if 302 Response)	Callback url
	State	No (if 302 Response)	state (Internal use)

Reference Implementation (OAuth 2.0 specifications [12])					
Name	**Implicit Authorization Request**	Method	GET	URL	/oauth2/authorize?response_type=&client_id=&redirect_uri=&state=
Params	● response_type (string) = token ● redirect_uri (string) [callback_url] ● client_id (string) ● state (string)				
Response	HTTP Status code	302 (Success) / Others (HTTP/1.1 error plus description)			
	Location	callback_url + access_token + state			

2.1.4.3 Resource Owner Password Credentials Grant

The resource owner password credentials grant workflow allows for the exchanging of the username and password of a user for an access token. When using the resource owner password credentials grant, the user provides the credentials (username and password) directly to the application. The application then uses the credentials to obtain an access token from the service.

Password Grant Request (OAuth 2.0 specification [11])			
Name	**Password Grant Token Request**		
Description	Requests an access token using the resource owner user/password.		
Primitive	**Password Grant Token Request**		
	Part name	**Optional**	**Description**
	Client ID	No	with the the client's ID (provided by IdM)
	Client Secret	No	with the client's secret (provided by IdM)
	grant_type	No	set to "password"
	username	No	with the user's username
	password	No	with the user's password
	scope	yes	with a space-delimited list of requested scope permissions
	Password Grant Token Response		
	Part name	**Optional**	**Description**
	Response/Error Code	No	200 Ok (If success)
	access token	No (if Success)	a JWT signed with the authorization server's private key
	token type	No (if Success)	string = "bearer"
	refresh token	No (if 302 Response)	an encrypted payload that can be used to refresh the access token when it expires.
	expiration time	No	with an integer representing the TTL of the access token
	Error Description	No	HTTP/1.1 error description

Reference Implementation (OAuth 2.0 specifications [12])					
Name	**Password Grant Token Request**	Method	POST	URL	/oauth2/token
Headers	● Authorization: Basic base64(client_id:client_secret) ● Content-Type: application/x-www-form-urlencoded				
Body	grant_type=password&username=[username]&password=[password]				
Response	HTTP Status code	200 (Ok) / Others (HTTP/1.1 error plus description)			

	JSON Object	{
		"access_token":["access_token"],
		"token_type":"bearer",
		"expires_in":[TTL in seconds],
		"refresh_token":["refresh_token"]
		}

2.1.4.4 Refresh Token Grant

Access tokens eventually expire; however, some grants respond with a refresh token which enables the client to refresh the access token. This method will provide a NEW access token, based on the credentials and scope requested by the original one (does NOT reactivate it).

Refresh Token Request (OAuth 2.0 specification [11])						
Name	**Refresh Token Request**					
Description	Request a new access token.					
Primitive	**Refresh Token Request**					
	Part name	**Optional**	**Description**			
	Client ID	No	with the the client's ID (provided by IdM)			
	Client Secret	No	with the client's secret (provided by IdM)			
	grant_type	No	set to "refresh_token"			
	refresh token	No	with the provided refresh token			
	scope	yes	with a space-delimited list of requested scope permissions			
	Refresh Token Response					
	Part name	**Optional**	**Description**			
	Response/Error Code	No	200 Ok (If success)			
	access token	No (if Success)	a JWT signed with the authorization server's private key			
	token type	No (if Success)	string = "Bearer"			
	refresh token	No (if 302 Response)	an encrypted payload that can be used to refresh the access token when it expires.			
	expiration time	No	with an integer representing the TTL of the access token			
	Error Description	No	HTTP/1.1 error description			
Reference Implementation (OAuth 2.0 specifications [12])						
Name	**RefreshToken Request**	Method	POST	URL	/oauth2/token	
Headers	• Authorization: Basic base64(client_id:client_secret) • Content-Type: application/x-www-form-urlencoded					
Body	grant_type=refresh_token&refresh_token=[refresh_token]					
Response	HTTP Status code	200 (Ok) / Others (HTTP/1.1 error plus description)				

	JSON Object	{
		"access_token":["access_token"],
		"token_type":"bearer",
		"expires_in":[TTL in seconds],
		"refresh_token":["refresh_token"]
		}				

2.1.4.5 Client Credential Grant

This grant is suitable for machine-to-machine authentication, for example for use in a cron job which is performing maintenance tasks over an API. Another example would be a client making requests

to an API that don't require user's permission. So, as there is no such thing as an 'authorizing user' because of the nature of this grant, this method only returns a token, not linked to any specific user.

<table>
<tr><th colspan="6">Client Credentials Token Request (OAuth 2.0 specification [11])</th></tr>
<tr><td>Name</td><td colspan="5">Client Credential Token Request</td></tr>
<tr><td>Description</td><td colspan="5">Request an access token, with no user's credentials.</td></tr>
<tr><td rowspan="11">Primitive</td><td colspan="5">Client Credential Token Request</td></tr>
<tr><td>Part name</td><td>Optional</td><td colspan="3">Description</td></tr>
<tr><td>Client ID</td><td>No</td><td colspan="3">with the the client's ID (provided by IdM)</td></tr>
<tr><td>Client Secret</td><td>No</td><td colspan="3">with the client's secret (provided by IdM)</td></tr>
<tr><td>grant_type</td><td>No</td><td colspan="3">set to "client_credentials"</td></tr>
<tr><td>scope</td><td>yes</td><td colspan="3">with a space-delimited list of requested scope permissions</td></tr>
<tr><td colspan="5">Client Credential Token Response</td></tr>
<tr><td>Part name</td><td colspan="2">Optional</td><td colspan="2">Description</td></tr>
<tr><td>Response/Error Code</td><td colspan="2">No</td><td colspan="2">200 Ok (If success)</td></tr>
<tr><td>access token</td><td colspan="2">No (if Success)</td><td colspan="2">a JWT signed with the authorization server's private key</td></tr>
<tr><td>token type</td><td colspan="2">No (if Success)</td><td colspan="2">string = "Bearer"</td></tr>
<tr><td>expiration time</td><td colspan="2">No</td><td colspan="2">with an integer representing the TTL of the access token</td></tr>
<tr><td>Error Description</td><td colspan="2">No</td><td colspan="2">HTTP/1.1 error description</td></tr>
<tr><td colspan="6">Reference Implementation (OAuth 2.0 specifications [12])</td></tr>
<tr><td>Name</td><td>Client Credentials Token Request</td><td>Method</td><td>POST</td><td>URL</td><td>/oauth2/token</td></tr>
<tr><td>Headers</td><td colspan="5">• Authorization: Basic base64(client_id:client_secret)
• Content-Type: application/x-www-form-urlencoded</td></tr>
<tr><td>Body</td><td colspan="5">grant_type=client_credentials</td></tr>
<tr><td rowspan="2">Response</td><td>HTTP Status code</td><td colspan="4">200 (Ok) / Others (HTTP/1.1 error plus description)</td></tr>
<tr><td>JSON Object</td><td colspan="4">{
"access_token":["access_token"],
"token_type":"bearer",
"expires_in":[TTL in seconds],
}</td></tr>
</table>

2.2 Components' validation: Functional and technical requirements

For each component and/or API required to support the identified interoperability points, this section will show the corresponding table with the functional and technical requirements to be assessed. It also documents the validation mechanisms, tools and processes followed to check the referred component features from the SynchroniCity requirements point of view. In the same way, it will describe the specific process designed to validate the referred interoperability point and/or framework component. Last but not least the metrics used to conduct the validation process of the corresponding technical requirement should be included as part of the table.

Technical and Functional requirements listed for each SynchroniCity component come directly from the previous analysis done in WP1, WP2 and WP3 and identified in the previous D4.1 [1].

2.2.1 Data models and NGSI interface

Data models to capture and represent context information are quite linked to the way this information is going to be managed through the NGSI interface. This is why the validation process for these two components includes both: it uses (and validates) NGSI interface to access the Data models, so, in

turn, checks them. This validation process also addresses the requirements listed in Table 2 and Table 3.

2.2.1.1 Addressed requirements (Data Models)

ID	TITLE	CATEGORY	TYPE
SR-INT-OPEN-01	Use of publicly accepted and open standards	Interoperability & Openness	Non-Functional
SR-LEGACY-01	Flexible support of new and legacy components	Legacy Compatibility & heterogeneous landscape	Non-Functional
SR-MODELS-01	Asset description taxonomies	Models	Functional
SR-MODELS-02	Standard and open data models	Models	Non-Functional

Table 2. Requirements addressed by SynchroniCity Data Models

2.2.1.2 Addressed Requirements (NGSI Interface)

ID	TITLE	CATEGORY	TYPE
SR-INT-OPEN-01	Use of publicly accepted and open standards	Interoperability & Openness	Non-Functional
SR-PERF-01	Real-time user experience	Performance	Non-Functional
SR-API-01	Standard and Open API	Data Management APIs	Non-Functional
SR-API-02	Publish/subscribe data channels	Data Management APIs	Functional
SR-API-03	Asset version management	Data Management APIs	Functional
SR-API-04	Resources status notification	Data Management APIs	Functional
SR-API-05	Lookup asset	Data Management APIs	Functional
SR-API-06	The system has to allow to access/consume data through RESTful API	Data Management APIs	Functional
SR-STORAGE-01	Physical data storage location	Data Storage Management	Non-Functional
SR-STORAGE-02	Categorization	Data Storage Management	Non-Functional

Table 3. Requirements addressed by NGSI Interface

2.2.1.3 Validation mechanisms

Each RZ is required to provide an implementation of the NGSI interface for Context Management described in section 2.1.2, that covers the set of defined primitives and supports the specified requirements for this Interoperability Point. Likewise, the context information offered through these interfaces must be compliant with the Data models and JSON-Schemas defined in Task 2.2. According to this, WP4 has defined and implemented a validation tool, the SynchroniCity data model validator (SC_DMV) [13].

First version of the SC_DMV is a CLI tool that validates a NGSI API, according to SynchroniCity NGSI reference implementation (NGSIv2) and the datasets (Context Data) following the data models defined within the SynchroniCity Project, available in the SynchroniCity GitLab. It should be run against a NGSIv2 compliant Context Manager and will provide a summary of its SynchroniCity compliance level. It is implemented as a command line Node program leveraging AJV JSON Schema Validator. SC_DMV offers the following checks:

- For Data models, it uses the SynchroniCity defined JSON Schemas (for each entity TYPE) to:
    - Validate that a certain entity TYPE has the required attributes. It does not mean that all the entities have them, but that the union of them does. It's a first quick check to obtain an overall approach of the compliance datasets level.
    - Validate that all entity TYPES existing in an Context Manager have the required attributes.

    - Validate that all entities of a certain TYPE follow the required schema (for a specified SynchroniCity compliant data model). It's a deeper analysis that returns the specific errors found for each entity belonging to each selected TYPE.
    - Validate that all entities in an Orion have the required schema, according to their TYPE. This identifies mismatching entities or possible new information sources.
- For NGSI Interface, that a certain URL exposes a valid NGSI API (NGSIv2 compliant):
    - Resource Directory functionalities
        - Entity creation (createEntity) [true/false] supports NGSI **registerContext** primitive
        - Entity query (readEntity and readEntityAttr) [true/false] supports NGSI **discoverContextAvailability** (specific Entity ID) primitive
        - Find entity by type (findEntityByType) [true/false] supports NGSI **discoverContextAvailability** (type parameter) primitive
        - Find entity by ID pattern (findEntityByIdpattern) [true/false] supports NGSI **discoverContextAvailability** (Entity ID modifier) primitive
        - Find entity by attribute value (findEntityByAttrvalue) [true/false] supports NGSI **discoverContextAvailability** (attributes parameter) primitive
    - Context Information management
        - Attribute entity query (readEntityAttr) [true/false] supports NGSI **queryContext** (specific Entity ID) primitive
        - Only value attribute entity query (readEntityAttrValue) [true/false] supports NGSI **queryContext** (specific Attr name) primitive
        - Update entity (updateEntity) [true/false] supports NGSI **updateContext** (append/update) primitive
        - Delete entity (deleteEntity) [true/false] supports NGSI **updateContext** (delete) primitive
    - Context Broker functionalities
        - Subscription creation (createSubscription) [true/false] supports NGSI **subscribeContextAvailability** and **subscribeContext** primitive
        - Subscription deletion (deleteSubscription) [true/false] supports subscription removal (not and NGSI primitive)

This tool allows different validation scenarios, considering one or several simultaneous checks. In all cases, the program returns a JSON file with the validation result, reporting the set of found datasets, which of them agree with SynchroniCity specifications and what is the list of errors found. It is intended for the RZ to help them polish their datasets, before a final report that will be included in this deliverable. This final report will reflect the availability of an NGSI proper interface and the available set of SynchroniCity datasets on each RZ. This tool evolves to offer different functionalities to the user, such as a REST interface that enables the integration of the tool with different user interfaces and wider tools.

2.2.2 Historical data API

According the Historical Data access interoperability point, this will provide historical data referred to a given entity and attribute within a specified timeslot, covering also requirements in Table 4. The SynchroniCity reference implementation for this API relies on a mongodb connected to an NGSIv2 compliant Context Manager. Although it is not specified the way this mongodb (historical data persistence layer) is later linked to the final API to read the data, most of the Reference Zones selected a modification of the FIWARE Short Term Historical data "Comet" [14] to implement this API. This implementation uses containers to be deployed and is based on NodeJS.

2.2.2.1 Addressed Requirements

ID	TITLE	CATEGORY	TYPE
SR-MODULARITY-01	Container technology	Decoupled & distributed components	Non-Functional
SR-INT-OPEN-01	Use of publicly accepted and open standards	Interoperability & Openness	Non-Functional
SR-SCALABILITY-01	Horizontal and vertical scaling	Scalability	Non-Functional
SR-PERF-02	Operational	Performance	Non-Functional
SR-API-01	Standard and Open API	Data Management APIs	Non-Functional
SR-API-05	Lookup asset	Data Management APIs	Functional
SR-API-06	The system has to allow to access/consume data through RESTful API	Data Management APIs	Functional
SR-STORAGE-01	Physical data storage location	Data Storage Management	Non-Functional
SR-STORAGE-02	Categorization	Data Storage Management	Non-Functional

Table 4. Requirements addressed by the SynchroniCity Historical data access API

2.2.2.2 Validation mechanisms

The SynchroniCity Historical Data Access specification consists on a REST API implementation, similar to the NGSI interoperability point. In fact, this historical API is based on the historical data extension proposed to the NGSI existing API by the new ETSI NGSI-LD draft standard [7], what will enable SynchroniCity interfaces be NGSI-LD compliant. For this reason, the SynchroniCity Data Model Validator was extended to support also the validation of this API according SynchroniCity framework reference implementation. For a given exposed valid SynchroniCity Historical API, it checks:

- Temporal (after, before, and between) query with "*lastn*" parameter.
    a. timerel=after & lastn=5
        i. [first, last] <- Get overall first and last values
        ii. data <- Get last 5 records after first
        iii. If size(data) is 0 or greater than 5 --> error: It returns no data or more than lastn
        iv. If newest(data) not equal to last --> error: Newest value is not the last
        v. data <- Get last 5 records after last
        vi. If size(data) greater than 1 --> error: It returns values after last
        vii. If size(data) is one and oldest(data) is not equal to last --> error: Oldest value is not last
    b. timerel=before & lastn=5
        i. [first, last] <- Get overall first and last values
        ii. data <- Get last 5 records before last
        iii. If size(data) is 0 or greater than 5 --> error: It returns no data or more than lastn
        iv. If oldest(data) not equal to last --> error: Oldest value is not the last
        v. data <- Get last 5 records before first
        vi. If size(data) greater than 1 --> error: It returns values after last
        vii. If size(data) is one and newest(data) is not equal to first --> error: Newest value is not first
    c. timerel=between & lastn=5
        i. [first, last] <- Get overall first and last values
        ii. data <- Get last 5 records after first
        iii. If size(data) is 0 or greater than 5 --> error: It returns no data or more than lastn
        iv. If oldest(data) not equal to last --> error: Newest value is not the last
- Only paging (limit and offset)

a. data1 <- Get with offset 0 and limit 2
b. If size(data1) greater than 2 --> error: Limit not working properly
c. data2 <- Get with offset 1 and limit 1
d. If data1[1] not equal to data2[0] --> error: Paging not working properly

- Temporal query (after, before, and between) with paging (limit and offset)
  a. timerel=after & paging
     i. [first, last] <- Get overall first and last values
     ii. data1 <- Get after first with offset 0 and limit 2
     iii. If size(data1) greater than 2 --> error: Limit not working properly
     iv. If newest(data1) not equal to first --> error: After not working properly with paging
     v. data2 <- Get after first with offset 1 and limit 1
     vi. If data1[1] not equal to data2[0] --> error: Paging not working properly
  b. timerel=after & paging
     i. [first, last] <- Get overall first and last values
     ii. data1 <- Get before last with offset 0 and limit 2
     iii. If size(data1) greater than 2 --> error: Limit not working properly
     iv. If newest(data1) not equal to first --> error: After not working properly with paging
     v. data2 <- Get after first with offset 1 and limit 1
     vi. If data1[1] not equal to data2[0] --> error: Paging not working properly
  c. timerel=after & paging
     i. [first, last] <- Get overall first and last values
     ii. data1 <- Get between first and last with offset 0 and limit 2
     iii. If size(data1) greater than 2 --> error: Limit not working properly
     iv. If newest(data1) not equal to first --> error: After not working properly with paging
     v. data2 <- Get after first with offset 1 and limit 1
     vi. If data1[1] *(element 1 of data array "data1")* not equal to data2[0] *(element 0 of data array "data2")* --> error: Paging not working properly

2.2.3 Security API

SynchroniCity architecture relies on OAuth 2.0 protocol (developed within the IETF OAuth Working Group [11]) to implement specific authorization flows for web applications, desktop applications, mobile phones, and IoT devices. SynchroniCity requests to all compliant frameworks to provide such an interface. The authorization server and the identity manager that supports this interface depends on each framework, and it is not mandatory to be the same, but should cover requirements in Table 5.

2.2.3.1 Addressed Requirements

ID	TITLE	CATEGORY	TYPE
SR-INT-OPEN-01	Use of publicly accepted and open standards	Interoperability & Openness	Non-Functional
SR-MONITORING-01	Data usage monitoring	Feedback and monitoring	Functional
SR-API-01	Standard and Open API	Data Management APIs	Non-Functional
SR-POLICY-01	Data and service access policy	Data Management APIs	Functional
SR-PRIVACY-01	Privacy policies guidelines	Data protection and Privacy	Functional
SR-PRIVACY-02	Data protection	Data protection and Privacy	Non-Functional
SR-PRIVACY-03	Anonymization	Data protection and Privacy	Non-Functional
SR-PRIVACY-04	Personal Data usage	Data protection and Privacy	Functional
SR-SECURITY-01	End-to-end secure communication	IoT infrastructure	Non-Functional
SR-SECURITY-03	Access policy	Platform	Functional

SR-SECURITY-04	Flexible security capabilities	Platform	Non-Functional

Table 5. Requirements addressed by SynchroniCity Security API

2.2.3.2 Validation mechanisms

SynchroniCity reference framework implementation proposes FIWARE KeyRock IdM [15] as the component that supports identity management and OAuth 2.0 API. At the time of validating and writing, most of the SynchroniCity reference zones are working on this interface, so there's no security API validation process properly defined. This is the reason why we prefer not to include this check within this document, but to detail it in D4.3: technical validation, phase 2.

2.2.4 Data marketplace

The SynchroniCity marketplace component [16] publishes within a catalogue datasets initially provided by RZs, featuring discovery and search functionalities so that resources can be easily and directly traded among users (e.g., data providers, application developers, researchers, etc.). By having an IoT data marketplace, application developers could obtain access to a multitude of sensors and/or actuators made available by assets owners to build new applications, thus enabling the creation of new smart city services. This component also covers the requirements listed in .

2.2.4.1 Addressed Requirements

ID	TITLE	CATEGORY	TYPE
SR-API-01	Standard and Open API	Data Management APIs	Non-Functional
SR-API-03	Asset version management	Data Management APIs	Functional
SR-API-04	Resources status notification	Data Management APIs	Functional
SR-API-05	Lookup asset	Data Management APIs	Functional
SR-LICENSE-01	Data licenses definition	Licence	Functional
SR-LICENSE-02	Customisable Licenses	Licence	Functional
SR-LICENSE-03	Pre-built Licenses	Licence	Functional
SR-SLA-01	SLA management	SLA	Functional
SR-SLA-02	SLA common metadata	SLA	Functional
SR-SECURITY-03	Access policy	Platform	Functional
SR-MODELS-01	Asset description taxonomies	Models	Functional
SR-PERF-01	Real-time user experience	Performance	Non-Functional
SR-MKTPLACE-01	Marketplace access	Marketplace	Functional
SR-MKTPLACE-02	Asset publication procedure	Marketplace	Functional
SR-MKTPLACE-03	Flexible revenue and pricing models	Marketplace	Functional
SR-MKTPLACE-04	Asset catalogue	Marketplace	Functional
SR-MKTPLACE-05	SynchroniCity compliance policy validation	Marketplace	Functional

SR-MKTPLACE-06	Asset request procedure	Marketplace	Functional
SR-MKTPLACE-07	Marketplace transparency	Marketplace	Non-Functional
SR-MKTPLACE-08	Marketplace peering	Marketplace	Functional

Table 6. Requirements addressed by SynchroniCity Data Marketplace

As internal functional building blocks, SynchroniCity marketplace component consists of the following elements:

- **Marketplace API** is the core element of the platform and includes several sub-components that expose a set of functions to interact with that. Specifically, the Marketplace API allows data providers to register or import data sources into the platform, and publish offerings containing its description (e.g., version, data model, endpoint URL), and allows data consumers (e.g., service developers) to discover and purchase offerings.
- **Marketplace portal** is an optional component, providing a user interface through which data providers and data consumers can interact with the platform, use the Marketplace API, and manage their accounts and information.
- **Security** includes three sub-components: (i) *Identity management* system to store and organize the identities of users; (ii) *Authentication* component that regulates the accesses to the platform; (iii) *Authorization* component that grants the access to the functionalities and digital assets according to predefined policies.

2.2.4.2 Validation Mechanisms

The validation mechanism for the SynchroniCity marketplace focuses on its core functionalities, enabled by its API [17], since its Marketplace web portal is supported by this API and the Security element is linked to the SynchroniCity Security components relying on the Oauth 2.0 interface to be validated on the Security API section.

Therefore, this validation covers the basic features of the marketplace, namely the functionalities that allow for managing data source specifications, data offerings (including the management of data licenses), and orders. To this aim, we have developed a validation script and integrated it into the SC_DMV tool described in [13]. The script performs a set of calls to the marketplace API and checks its behavior as detailed in the following.

- **Data Source Specification API**
  - *List Data Source Specifications*: validate that all the data source specifications registered in the marketplace are properly retrieved.
  - *Create Data Source Specification*: validate that a new data source specification is registered in the marketplace.
  - *Get Data Source Specification*: validate that a data source specification with a certain `id` is properly retrieved.
  - *Update Data Source Specification*: validate that a data source specification with a certain `id` is properly updated with certain parameters.

- **Data Offering Management API**
  - *List Data Offerings:* validate that all the data offerings published in a catalog having a certain `catId` are properly retrieved.

- *Create Data Offering*: validate that a new data offering is published in a catalog having a certain `catId`, with all the parameters specified, including the `licenseSpecification`, which defines the data license associated with the data offering.
- *Get Data Offering*: validate that a data offering with a certain `id` is properly retrieved from a catalog having a certain `catId`.
- *Update Data Offering*: validate that a data offering with a certain `id` is properly updated with certain parameters.

- **Order Management API**
  - *List Orders:* validate that all the orders added to the cart are properly retrieved.
  - *Create Order*: validate that a new order from a consumer and for a certain data offering is properly added to the cart.
  - *Get Order*: validate that an order with a certain `id` is properly retrieved from the cart.
  - *Update Order*: validate that an order with a certain `id` is properly updated with certain parameters.

3 Reference Framework component and APIs validation per Reference Zone deployment

Once the key components of the SynchroniCity compliant framework have been introduced and their corresponding validation process described, this section summarizes the results obtained from the different validation scenarios executed in each RZ's deployed framework, according to SynchroniCity requirements and specifications. The outcomes of these validation processes will provide a real and quick look about the SynchroniCity compliance on each RZ.

The validation process for phase one has focused on data readiness, this is, NGSI interface, to access context information, SynchroniCity defined Historical API, to obtain the related historical information and the so far defined SynchroniCity data models. It also includes the SynchroniCity Marketplace validation, were it is available, in order to demonstrate its functionalities and foster its adoption.

Each Reference Zone has collaborated in this validation process by providing the required accesses to deployed components. WP2 (Task 2.1) has collected and updated the different endpoints linked to the available interfaces on cities frameworks, as well as the present and shared data models. The endpoints referred to NGSI interfaces, SynchroniCity defined Historical API (not those exposing directly other historical interfaces, such STH Comet or Quantum Leap) and the set of uploaded data models are considered within WP4 to execute the corresponding validation mechanism described in the previous section.

According to this, the phase one validation process will show the corresponding reference zones' endpoints exposing components (interfaces) that support above mentioned interoperability points. The result of this process will be shown following this colour code (Table 7):

	The component/Interface satisfactory passed the corresponding validation process, with very minor remarks (if any).
	The component/Interface is present/reachable but failed the corresponding validation process because of required mismatched functionalities and/or not SynchroniCity compliant. In any case, these reported issues are feasible to be rectified.
	The component/interface has not been validated because external issues (e.g. not reachable, need specific credentials that have not been provided, etc.)
	This component/interface is not considered in Validation Process Phase I

Table 7. Colour code showing validated component result/status

A deeper description of IoT deployments and SynchroniCity available components on each RZ can be found in D2.8 [18]

3.1 Porto's Reference Zone framework implementation and instance validation

The city of Porto pioneered the idea of a living lab, fostering a wide range of projects that emerged from the University and R&D institutions, focusing on the development of a consistent infrastructure to promote data driven decision making and effective governance support. This is supported by the city's IoT infrastructure, which includes the following:

- Environmental sensor network (ESN): noise, meteorological parameters and air quality sensing stations (±30 units);

- Beacons: active (BLE, ±1.000 units) and passive (NFC, ±1.000 units);
- Traffic monitoring: vehicle counters (±120 units)

In terms of SynchroniCity approach, the data referenced above is made available through the FIWARE Orion Context Broker and the Comet Short Time Historic API, using the standard data models made available by FIWARE (where applicable).

3.1.1 Porto's list of available endpoints, interfaces and components

PORTO's Reference Zone SynchroniCity Framework INSTANCE				
Interfaces (Interoperability Points)	**Endpoint**	**Description**	**Auth. Required**	**Reference standards / Specifications**
Context Information Management	https://broker.fiware.urbanplatform.portodigital.pt	Orion Context Broker endpoint	NO	NGSI Interface: NGSIv2
Identity Management	http://idm.urbanplatform.portodigital.pt/	IdM Keyrock	YES	OAuth 2.0
Historical Data	https://ql.urbanplatform.portodigital.pt/	QuantumLeap REST API implementation	NO	SynchroniCIty Historical API

Table 8. Porto's deployed components (Synchronicity Architecture)

3.1.2 NGSI interface & Data models

Using SynchroniCity data model validator (SC_DMV) tool outcomes, this is the list (Table 9) of available data models in Porto's Context Manager component, through the corresponding NGSI interface and according SynchroniCity defined data models.

RZ Data available through context broker API						
Entity	**Description**	**Original Data Model**	**SynchroniCity Data Model**	**Notes**	**Nº of Entities**	
					Present	**Valid**
NoiseLevelObserved	Noise Level	NoiseLevelObserved	Environment (NoiseLevelObserved)		9	9
AirQualityObserved	Air quality data	AirQualityObserved	Environment (AirQualityObserved)		1	1
WeatherObserved	Meteorological parameters	WeatherObserved	Weather (WeatherObserved)		7	1
Vehicle	Vehicle location and speed	Vehicle	Vehicle (Vehicle)	Not yet available	246	0
TrafficFlowObserved	Vehicle count	TrafficFlowObserved	Transportation (TrafficFlowObserved)		210	0
PointOfInterest	Points of interest (POIs) data	PointOfInterest	Point of Interest (PointOfInterest)		2545	0

Event	Events data	Event	Event (Event)	Not yet exploitable	622	622
WeatherForecast	Weather forecast service	WeatherForecast	Weather Forecast (Weatherforecast)		115	115
Device	Device	Device	Device		211	0

Table 9. Porto's list of NGSI available context entities (present vs validated)

3.2 Santander's Reference Zone framework implementation and instance validation

Santander Internet of Things facility is based on a real IoT deployment in an urban setting. It encompasses IoT deployments in different key areas of the city infrastructure, ranging from public transport, logistics facilities, public places and buildings, environment, workplaces and residential areas, thus creating the basis for development of a Smart City. This deployment exhibits the diversity, dynamics and scale that are essential for the evaluation of advanced protocol solutions.

Santander's SynchroniCity framework uses NGSI specifications as Northbound and southbound API interfaces to manage context information, providing its own instance of the FIWARE Orion Context Broker. Security Framework will be also based on Oauth2.0 protocol. Santander gets its own instance of the classic FIWARE security set, including KeyRock IdM, Wilma PEP proxy and AuthZForce PDP. Santander SynchroniCity Instance will also support FIWARE Cygnus connector for long term historical data. Santander provides access to historical data using SynchroniCity Historical API specification.

Aligned also with the project's framework, Santander RZ deploys its own instance of the SynchroniCity Marketplace, integrated with the security framework, the context manager and the historical data access, to provide a clear management interface to access Santander's smart city resources.

3.2.1 Santander's list of available endpoints, interfaces and components

SANTANDER's Reference Zone SynchroniCity Framework INSTANCE				
Interfaces (Interoperability Points)	**Endpoint**	**Description**	**Auth. Required**	**Reference standards/Specifications**
Context Information Management	https://context.san.synchronicity-iot.eu/	Orion Context Broker endpoint	YES	NGSI Interface: NGSIv2
Identity Management	https://auth.san.synchronicity-iot.eu/	IdM Keyrock	YES	OAuth 2.0
Historical Data	https://historical.san.synchronicity-iot.eu/	Cygnus+Mongo based implementation	YES	SynchroniCIty Historical API
Marketplace	https://marketplace.san.synchronicity-iot.eu/	SynchroniCity implementation	YES	TM Forum APIs

Table 10. Santander's deployed components (Synchronicity Architecture)

3.2.2 NGSI interface & Data models

Using SynchroniCity data model validator (SC_DMV) tool outcomes, this (Table 11) is the list of available data models in Santander's Context Manager component, through the corresponding NGSI interface and according SynchroniCity defined data models.

RZ Data available through context broker API						
Entity	**Description**	**Original Data Model**	**SynchroniCity Data Model**	**Notes**	**Nº of Entities**	
					Present	**Valid**
AirQualityObserved		AirQualityObserved	Environment (AirQualityObserved)		65	65
Beach		Point of Interest (Beach)	Point of Interest (Beach)		13	13
BikeHireDockingStation		BikeHireDockingStation	Transportation (BikeHireDockingStation)		17	17
BusArrivalEstimation		BusArrivalEstimation	UC-Transportation (BusArrivalEstimation)		982	982
BusStop		BusStop	UC-Transportation (BusStop)		446	446
Device		Device (Device)	Device (Device)		283	283
GreenspaceRecord		GreenspaceRecord	Parks & Gardens (GreenspaceRecord)		18	18
Museum		Point of Interest (Museum)	Point of Interest (Museum)		10	10
NoiseLevelObserved		NoiseLevelObserved	Environment (NoiseLevelObserved)		19	19
ParkingSpot		ParkingSpot	Parking (ParkingSpot)		262	262
OnStreetParking		OnStreetParking	Parking (OnStreetParking)		23	19
PointOfInterest		PointOfInterest	Point of Interest (PointOfInterest)		214	214
PointOfInterest:shop		PointOfInterest	Point of Interest (PointOfInterest)		2170	2170
TrafficFlowObserved		TrafficFlowObserved	Transportation (TrafficFlowObserved)		465	311
WeatherObserved		WeatherObserved	Weather (WeatherObserved)		181	181
busLine		busLine	UC-Transportation (busLine)		35	35

Table 11. Santander's list of NGSI available context entities (present vs validated)

3.2.3 Data Marketplace

For the Marketplace validation, the SynchroniCity data model validator has been extended to execute the component's validation process defined in previous section. It executes, so, the listed

HTTP requests against the given Marketplace endpoint to check what they return. These (Table 12) are the results obtained for Santander's Marketplace endpoint.

SANTANDER's Marketplace INSTANCE				
API	**Method**	**Request**	**Expected Result**	**Valid**
Data Source Specification	GET	List data source specification	Retrieve all data source specifications	Yes
Data Source Specification	POST	Create data source specification	Create a new data source specification	Yes
Data Source Specification	GET	Get data source specification	Retrieve specific data source specification with the given *id*	Yes
Data Source Specification	PUT	Update data source specification	Update specific data source specification with the given *id*	Yes
Data Offering Specification	GET	List data offering specification	Retrieve all data source specifications	Yes
Data Offering Specification	POST	Create data offering specification	Create a new data offering specification	Yes
Data Offering Specification	GET	Get data offering specification	Retrieve specific data offering specification with the given *id*	Yes
Data Offering Specification	PUT	Update data offering specification	Update specific data offering specification with the given *id*	Yes
Order management	GET	List orders	Retrieve all orders added to cart	Yes
Order management	POST	Create order	Create a new order	Yes
Order management	GET	Get order	Retrieve specific order the given *id*	Yes
Order management	PUT	Update order	Update specific order with the given *id*	Yes

Table 12. Santander's Marketplace component validation

3.3 Antwerp's Reference Zone framework implementation and instance validation

The Antwerp RZ shares data from different IoT projects, managed by different entities within the reference zone: the City of Antwerp Smart Zone. This is a new and important initiative that has the integration of IoT/IoE in our everyday life as a main goal. The current IoT services are focused on multiple domains: Smart Logistics (e.g. parking slots), Smart Mobility & Traffic (e.g. Cycling Push Sensors, Bike sharing stations and Camera feed - traffic info), Smart Environmental monitoring (e.g. fixed and mobile air quality sensors and weather stations) and Smart Waste (e.g. smart waste sorting litter bins and containers on the street for recycling). Antwerp RZ offers its data catalogue mainly through two ways: own API Marketplace and Static Open Data. In terms of SynchroniCity framework, Antwerp's datasets will be expanded and adapted to the agreed Synchronicity data models and while implementing the corresponding adaptors. In order to provide NGSI support, Antwerp RZ will implement its own instance of the FIWARE Orion Context Broker. The Security Framework of the Antwerp RZ will be integrated within SynchroniCity's one, according to Synchronicity security workgroup guidelines. For Historical data management, the city will proceed according to

Synchronicity architecture requirements, in order to integrate with Synchronicity Historical Data approach.

3.3.1 Antwerp's list of available endpoints, interfaces and components

ANTWERP's Reference Zone SynchroniCity Framework INSTANCE				
Interfaces (Interoperability Points)	**Endpoint**	**Description**	**Auth. Required**	**Reference standards/Specifications**
Context Information Management	https://ext-api-gw-p.antwerpen.be/sirus/contextmanagement-a/v2	Orion Context Broker endpoint	YES	NGSI Interface: NGSIv2
Identity Management		WSO2 based	YES	OAuth 2.0
Historical Data		Based on STH Comet		

Table 13. Antwerp's deployed components (Synchronicity Architecture)

3.3.2 NGSI interface & Data models

Using SynchroniCity data model validator [13] tool outcomes, this is the list (Table 14) of available data models in Antwerp's Context Manager component, through the corresponding NGSI interface and according SynchroniCity defined data models.

RZ Data available through context broker API						
Entity	**Description**	**Original Data Model**	**SynchroniCity Data Model**	**Notes**	**Nº of Entities**	
					Present	**Valid**
WasteContainer			WasteContainer		127	107

Table 14. Antwerp's list of NGSI available context entities (present vs validated)

3.4 Eindhoven's Reference Zone framework implementation and instance validation

The Eindhoven IoT deployment includes data from different IoT projects and/or living labs, managed by different entities within the city ecosystem. This way, it manages data related to traffic flows, air quality, noise, bikes sharing and pedestrian counting. In terms of the SynchroniCity framework, Eindhoven will use an own instance with stable and integrated versions of the IDAS IoT/Device Manager and Agents (on MongoDB), the Orion Context Broker, the Proton Complex Event Processing and the Cygnus Connector to PostgreSQL CKAN-based data storage with (linking and/or harvesting to/from) the Eindhoven Open Data Portal based on OpenDataSoft tooling and DCAT-AP Data Catalog Vocabulary [19]. Eindhoven will also use the uniform historic data API to store historical IoT data in the CKAN data store.

3.4.1 Eindhoven's list of available endpoints, interfaces and components

EINDHOVEN's Reference Zone SynchroniCity Framework INSTANCE				
Interfaces (Interoperability Points)	**Endpoint**	**Description**	**Auth. Required**	**Reference standards/Specifications**
Context Information Management	http://212.159.228.70:1026	Orion Context Broker endpoint	NO	NGSI Interface: NGSIv2
Identity Management				
Historical Data				

Table 15. Eindhoven's deployed components (Synchronicity Architecture)

3.4.2 NGSI interface & Data models

Using SynchroniCity data model validator (SC_DMV) tool outcomes, this is the list (Table 16) of available data models in Eindhoven's Context Manager component, through the corresponding NGSI interface and according SynchroniCity defined data models.

RZ Data available through context broker API						
Entity	**Description**	**Original Data Model**	**SynchroniCity Data Model**	**Notes**	**Nº of Entities**	
					Present	**Valid**
AirQualityObserved	Real time air quality data in Eindhoven RZ	AirQualityObserved	Environment (AirQualityObserved)	Validated through WP4 Validator	35	35

Table 16. Eindhoven's list of NGSI available context entities (present vs validated)

3.5 Helsinki's Reference Zone framework implementation and instance validation

IoT infrastructure in Helsinki is taking form as devices are being deployed in the city. Devices are generally operated by the organizations using the data in their applications. The data is planned to be imported into the Synchronicity platform via custom software agents that subscribe to the data sources. This presupposes that all data that we wish to import to the Synchronicity platform must be provided openly by the respective organizations. The Helsinki RZ framework is based on the integration of these RZ data sources with the Digitransit [20] platform via the NGSIv2 API [9] in pursue of a large-scale piloting facility. Available data sources provide measurements via sensors' own proprietary protocols, typically over HTTP or MQTT on 3G networks. These sources are read in via Sensor agents, parsing the formats and storing the data both in a fast memory cache for real time use, and to a database for historical searches. Then, Northbound APIs provide this data via NGSIv2 API, both for the RZ Pilot use case and as general Open Data.

3.5.1 Helsinki's list of available endpoints, interfaces and components

HELSINKI's Reference Zone SynchroniCity Framework INSTANCE				
Interfaces (Interoperability Points)	**Endpoint**	**Description**	**Auth. Required**	**Reference standards/Specifications**
Context Information Management	https://synchronicity.cs.hut.fi/orion/	Orion Context Broker endpoint	NO	NGSI Interface: NGSIv2
Identity Management				
Historical Data				

Table 17. Helsinki's deployed components (Synchronicity Architecture)

3.5.2 NGSI interface & Data models

Using SynchroniCity data model validator [13] tool outcomes, this is the list (Table 18) of available data models in Helsinki's Context Manager component, through the corresponding NGSI interface and according SynchroniCity defined data models.

RZ Data available through context broker API						
Entity	**Description**	**Original Data Model**	**SynchroniCity Data Model**	**Notes**	**Nº of Entities**	
					Present	**Valid**
BikeHireDockingStation	Helsinki area city bikes	bikeRentalStation (Digitransit)	Transportation (BikeHireDockingStation)		255	255

Table 18. Helsinki's list of NGSI available context entities (present vs validated)

3.6 Manchester's Reference Zone framework implementation and instance validation

In order to provide NGSI support, as Synchronicity requirement, Manchester RZ will implement its own instance of the FIWARE Orion Context Broker. As Manchester data is not currently in FIWARE standards, the City will identify a list of core datasets from the Manchester platforms including all live data; Digital catapult will provide the conversion scripting from the Manchester BT data hub [18] to FIWARE data models. Where these aren't available, it is following common practice around common data fields. Security and historical data will be in line with the Synchronicity framework once the Manchester local deployment is in place.

3.6.1 Manchester's list of available endpoints, interfaces and components

MANCHESTER's Reference Zone SynchroniCity Framework INSTANCE				
Interfaces (Interoperability Points)	**Endpoint**	**Description**	**Auth. Required**	**Reference standards/Specifications**
Context Information Management	http://217.172.12.160:1026/	Orion Context Broker endpoint	NO	NGSI Interface: NGSIv2

Identity Management				
Historical Data				

Table 19. Manchester's deployed components (Synchronicity Architecture)

3.6.2 NGSI interface & Data models

Using SynchroniCity data model validator [13] tool outcomes, this (Table 20) is the list of available data models in Manchester's Context Manager component, through the corresponding NGSI interface and according SynchroniCity defined data models.

RZ Data available throw context broker API						
Entity	**Description**	**Original Data Model**	**SynchroniCity Data Model**	**Notes**	**Nº of Entities**	
					Present	**Valid**
AirQualityObserved			AirQualityObserved		67	47
BikeHireDockingStation			BikeHireDockingStation		1	1
CycleCounting				Datamodel under discussion	1	0
NoiseLevelObserved			Environment (NoiseLevelObserved)		6	6
OffStreetParking			OffStreetParking		1	1
OnStreetParking			OnStreetParking		40	0
PedestrianCounting				Datamodel under discussion	8	0
WeatherObserved			Weather (WeatherObserved)		6	6

Table 20. Manchester's list of NGSI available context entities (present vs validated)

3.7 Milan's Reference Zone framework implementation and instance validation

In 2017 the Municipality of Milano RZ started a process to define a standard to deploy applications on its legacy systems and platforms that should be completed by the end of 2018. This will conform the Milan's Smart City infrastructure [18]. In order to provide NGSI support, as Synchronicity requirement, Milano RZ needs to re-expose on its APIStore the already existing APIs involved in Synchronicity project into the NGSI format according the agreed Synchronicity data models. These newly APIs will publish datasets according the agreed NGSI data models. Milan RZ implements the OAuth2 protocol for Authorization based on an instance of the KeyManager module of the WSO2 [21] platform. The Authentication layer will be implemented in the very next future by a WSO2 Identity Manager module. For Historical data management, since changing data-models already in use for storage could severely impact Municipality processes, on the first step Milano RZ need to create connectors to access to already existing data in order to expose them into NGSI protocols. A data-model changing process can be evaluated after the success of SynchroniCity project and a costs

and benefits evaluation. For newly generated data-models as result of the SynchroniCity project, Milano RZ will proceed to historicize according to SynchroniCity NGSI data model.

3.7.1 Milan's list of available endpoints, interfaces and components

MILAN's Reference Zone SynchroniCity Framework INSTANCE				
Interfaces (Interoperability Points)	**Endpoint**	**Description**	**Auth. Required**	**Reference standards/Specifications**
Context Information Management	https://api.comune.milano.it/synchronicity/context/1.0	Orion Context Broker endpoint	YES	NGSI Interface: NGSIv2
Identity Management		WSO2 based	YES	OAuth 2.0
Historical Data	https://api.comune.milano.it/synchronicity/historical/1.0	Cygnus+Mongo based implementation	YES	SynchroniCIty Historical API

Table 21. Milan's deployed components (Synchronicity Architecture)

3.7.2 NGSI interface & Data models

Using SynchroniCity data model validator (SC_DMV) tool outcomes, this (Table 22) is the list of available data models in Milan's Context Manager component, through the corresponding NGSI interface and according SynchroniCity defined data models.

RZ Data available throw context broker API						
Entity	**Description**	**Original Data Model**	**SynchroniCity Data Model**	**Notes**	**Nº of Entities**	
					Present	**Valid**
BikeHireDockingStation			BikeHireDockingStation		3	0
WeatherObserved	Meteorological data (e.g. temperature, humidity) coming from Weather Sensors (LAMPPOST API)	LAMPPOST legacy model	Weather (WeatherObserved)	The mapping was achieved through the use of Visual Mashup Editor tool	7	0

Table 22. Milan's list of NGSI available context entities (present vs validated)

4 Results and Conclusions

The Table 23 summarises the outcomes of the Phase I validation process, showing which are the current SynchroniCity framework components properly deployed on each Reference Zone, as well as what are the plans of each of them regarding this deployment.

At the time of writing this document, endpoints from Carouge have not been provided for validation. The access keeps internal to Carouge's RZ and so the validation process. The results will be shown in Phase 2 (D4.3).

Validation TESTS (1st ROUND - May/July 2018)			Last Update on: 13/07/2018						
Reference Framework Implementation			SynchroniCity Reference Zones						
			Porto	Santander	Antwerp	Carouge	Eindhoven	Helsinki	Manchester	Milan
Basic	NGSI Interface		Orion CB	Orion CB	Orion CB	Orion CB	Orion CB	Orion CB	Orion CB	Orion CB
	Security	OAUTH2.0	IdM KR	IdM KR	WSO2 IdM	IdM KR				WSO2 IdM
		PeP-Proxy	Wilma Plus	Wilma Plus		Wilma				
	Historical Data Access	Sync API (long Term)	Quantum Leap	Cygnus/Mongo/Nodejs		Cygnus+ Mongo				Cygnus/ Mongo
		Short Term	Comet		Comet					Comet
Advanced	Digital Single Market			Sync. Marketplace	Kong Market					
	IoT Agents		IDAS			UDG MPI				

Component validated (reachable and operational)

Partial validation

Planned (RZ plans to deploy it but no endpoint still available)

Not covered in Phase 1 validation process

Table 23. Validated SynchroniCity components per RZ

These validation results derive on the NGSI interface interoperability point implementation achieved on all Reference Zones through the FIWARE Orion Context Broker. This ensures a common way of context information management, based on the NGSIv2 protocol that enables both, southbound and northbound accesses to update and retrieve context data, as well as valuable mechanisms to create context entities, discover and subscribe context data. This is also aligned with the incoming ETSI NGSI-LD standard to manage context, what makes this first SynchroniCity goal a valuable point to progress towards the common IoT framework.

At the same line, several Reference Zones have adopted the SynchroniCity proposed Historical API to retrieve historical data from context entities. Three out of eight cities are already providing these methods to obtain historical series plus another one that is currently working on this implementation. This Historical API complements the NGSIv2 specification, used by all RZs, to fulfil the NGSI-LD functionalities. The rest of the cities just notified their intentions to proceed in a similar way.

This document also defined the methods and process to validate the SynchroniCity's security API to identify and authenticate end users. This is based on OAuth 2.0 standard and supported by each Reference Zone Identity Manager/Security Server. Due to the early stage of these components on most of the reference zones and the requirements in terms of identities and applications creations needed to properly check the OAuth 2.0 flow on each framework instance, this validation will be

included in Phase 2 deliverable. However, the requests needed to validate the NGSI interface in some of the implementations allowed the additional checking of the Policy Enforcement Point (PeP), component not initially required as an interoperability point but which enables the identification and authorization of a context consumer. In this case, this component was implemented by an extension of the FIWARE Wilma PeP Proxy, called Wilma Plus and provided by DigiCat. This enabler was verified in two of the reference zones. The rest haven't provided yet this component.

As an extra procedure within this Phase 1, the SynchroniCity Marketplace validation process has also been introduced and executed in Santander RZ. Although this is not a mandatory component to be deployed on each Reference Zone (some of them already have a solution for this), the marketplace is a valuable component to show, share and manage IoT assets and data sources, so it is worth checking the available instance to show this component's capabilities and foster its usage among those cities that already don't have this solved. The SynchroniCity Marketplace is being evolved in close contact with FIWARE technical partners, so it is expected to become a FIWARE compliant enabler in the near future.

On the other hand, Table 23 shows that further progress on components and interoperability points beyond the NGSI interface is needed to converge on a real common framework. In this sense, the security layer urgently needs to be deployed and integrated on most of the Reference Zones instances in order to guarantee the proper access to context resources and manage them before facing the results of the SynchroniCity Open Call. Also, all the historical series represent a valuable information source to implement IoT solutions, and, although SynchroniCity has defined a simple and easy to implement Historical API, this is still needed to be deployed in most of the cities. Progresses on these lines are expected during the SynchroniCity Open Call time slot, so, for Validation Phase 2, all SynchroniCity frameworks will be ready on all participating cities.

Another important interoperability point refers to Datasets and data models available on each framework. Table 24 summarises the current status on the shared datasets on each Reference Zone. This snapshot considers only those datasets accessible through the given RZ's NGSI interface and validates it using the agreed SynchroniCity data models.

Validation TESTS (1st ROUND - May/July 2018)		***Last Update on:***							***13/07/2018***
		Num. of Valid Entities reported (according Sync Data models)							
Data Models		**SyncroniCity Reference Zones**							
Data Sets	Data Schema	Porto	Santander	Antwerp	Carouge	Eindhoven	Helsinki	Manchester	Milan
Bike Hire Docking Stations	BikeHireDockingStation		17				255	1	3 (all fail)
Air Quality Observed	AirQualityObserved	1	65			35		67 (20 fail)	
Parking (OffStreet)	OffStreetParking							1	
Parking (OnStreet)	OnStreetParking		23 (4 fail)					40 (all fail)	
Parking Spot	ParkingSpot		262						
Noise Level Observed	NoiseLevelObserved	9	19						
Weather Observed	WeatherObserved	7 (6 fail)	181					6	7 (all fail)
Weather Forecast	WeatherForecast	115							
Traffic Flow Observed	TrafficFlowObserved	210 (all fail)	465 (154 fail)						
Bus Line	SAN Bus Datamodels		35						

Bus Stop	SAN Bus Datamodels		446						
Bus Timetables Real Time (estimations)	SAN Bus Datamodels		982						
Point Of Interest	PointOfInterest	2545	2394						
Alert	Alert								
City Event	Event	622							
Public Transport Fleet Vehicles	Vehicle	246 (all fail)							
Civic Issue Tracking	Open311: ServiceRequest								
Bicycle Tracking	TrafficFlowObserved								
Waste Containers Assets	Waste Container Model			127 (20 fail)					
Parks & Gardens	Green Space Record		18						
Device	Device	211 (all fail)	283						
Beach	Beach		13						

Partial validation (not all data models complies)

Full validation

Table 24. Available SynchroniCity compliant datasets

These SynchroniCity compliance datasets available on each SynchroniCity framework directly represents the project capability to support valuable and replicable end user applications. As all SynchroniCity Reference Zones MUST use these agreed data schemas to update and share their context resources, the more cities using the same data model, the more replicable would be a solution based on them, and, as these data models should be the baseline to define the SynchroniCity services and final applications, the more available datasets, the more versatile and functional they would be. According to this, these SynchroniCity compliant datasets is the most relevant point to improve during this Open Call duration, in order to get the widest range of IoT resources available for the Open Call applications development.

In summary, SynchroniCity is an appealing ecosystem which is well reflecting what is happening in the digital transformation of worldwide cities. The adoption of well stablished components, enablers and APIs takes place at a different pace. At time being, as envisioned in the Description of Action, there are a some of RZs which are essentially ready to support the stress imposed by the open call pilots. From now till the moment when experiments will start its execution, we expect that most of the RZs will have integrated the assets and components which will foster replication.

4.1 Validation process assessment

As described in Section 2.2, the different validation processes carried out to check required components' functionalities consist on a specified set of HTTP calls (since all SynchroniCity interoperability points supporting a framework component exposes a REST interface) plus the evaluation of their responses. As both: the method calls, and the expected responses has been listed and well defined within WP2 (expressed on the SynchroniCity framework reference implementation

and in the SynchroniCity data models), these validation processes have been synthesized by WP4 into a tool, initially known as SynchroniCity data model validator (13).

On its first version, the SC_DMV tool was intended as a command-line interface (CLI) to query for specific data entities through the NGSI interface, validating the NGSI API on a given context manager endpoint and the data models available through it. As the validation processes evolve, this tool upgraded to support other APIs and implement a REST interface to enable its integration on incoming SynchroniCity tools and user interfaces. This tool is documented and downloadable in the SynchroniCity GitLab [13]. Distributed as open source, current version of the tool supports:

- Command Line and REST interfaces, enabling the use of authentication/identification headers, needed to access some of the components' instances using the OAuth 2.0 protocol
- NGSI API and that data in instances of Orion Context Broker (NGSIv2 specifications)
- Data models defined within the SynchroniCity Project. It leverages AJV JSON Schema Validator. Supported SynchroniCity models are listed in [6]. Each time a new data model is approved within SynchroniCity framework, its schema is added to the tool.
- Historical API defined in the SynchroniCity Project

SC_DMV has been thought as an evolving tool, guided by the WP4 validation process that will define its requirements and functionalities. In this sense, new versions will appear to support new features and components as they are required. Next SC_DMV version will include:

- OAUTH2 interface validation, to cover the defined OAuth 2.0 flows
- Marketplace APIs validation integration, that will add the calls to properly validate an instance of the SynchroniCity marketplace

The WP4 overall validation process (Phase 1) provides, as a main outcome, a snapshot of the deployment status of each SynchroniCity framework on RZs, as well as the compliance of the available datasets with the SynchroniCity data models. Aligned to this, the SC_DMV tool has been designed to support a twofold functionality:

- In a punctual use, it covers the main technical validation background, providing the results of validating each of the involved (in Phase 1) interoperability points and interfaces. This way is easier, for the WP4 team, to obtain the deployment status of each reference zone and assess on the improvements needed.
- In an iterative way, the tool can be used by the reference zones to check both, the status of a single component or the overall deployed set, according the SynchroniCity requirements. In this sense, it also allows the compliance checking of the available context information, so, the reference zone can get a clear picture about how to proceed to achieve a complete SynchroniCity compliance on its framework. Here, WP4 team will assess, together with WP2 and WP3, on the different options to evolve the deployed components and data models to get valid.

The process of validation will continue, to complete phase II and beyond, till all the reference zones have finalised all their deployments: Reference Zones will set up new components and data models running their corresponding validation process assessed by WP4 and, in turn, periodically WP4 will execute the whole process on all available endpoints to have an updated view of the overall status.

4.2 KPIs

Key Performance Indicators (KPIs) are an important consideration in defining a comprehensive validation as they provide information on how a particular process or product is progressing. In the case of WP4 technical validation, they provide a quick overview of how SynchroniCity framework deployment on RZs is evolving in time. The KPIs presented in this document will be also revisited

and expanded in phase II, contributing to D4.3 (M27), to show the evolution of the technical framework of the project and, in particular, of the different RZs.

From the list of Project's KPIs described in [1], two of them (Table 25) are specifically related to SynchroniCity framework technical validation.

KPI	Description	Indicator unit	Expectations
IoT connected devices	Number of IoT connected devices implemented during the project lifecycle in all the pilots	#	10000
	Total number of IoT connected devices by the end of the project, including previously installed	#	100000
Open data sets	Number open data sets in use, in all the pilots	#	70

Table 25. Project KPIs related with the technical validation (extracted from D4.1)

Taking them as a baseline, a set of KPIs (Table 26) has been defined for measuring the evolution of the technical framework.

#	KPI	Description	Indicator unit	Measurement
1	SynchroniCity data sets	Number of available data sets according to SynchroniCity data models	#	Total and per RZ
2	Data sets shared	Number of data sets that are shared by 2 or more RZs	#	Total
3	Entities	Number of available entities (through NGSI interface)	#	Total and per RZ
4	SynchroniCity cities	Number of RZs with a NGSI interface exposing data sets according to SynchroniCity data models	#	Total

Table 26. Technical validation (phase I) KPIs

Based on the validation results presented in Table 23 and Table 24, 7 of the 8 core pilot cities have an open NGSI interface to expose SynchroniCity data sets. These cities are exposing 19 different data sets that have been validated according to SynchroniCity data models, where 5 of them are common to 2 or more RZs. The total number of entities that were validated in all the RZs reaches to 8791.

Table 27 and Table 28 present the value of the technical validation KPIs, in global values and divided per RZ, respectively, available in M19.

#	KPI	Description	Value
1	SynchroniCity data sets	Number of available different data sets according to SynchroniCity data models (Including those duplicated in several RZs)	19 (28)
2	Data sets shared	Number of data sets that are shared by 2 or more RZs	5
3	Entities	Number of available entities (through NGSI interface)	8791
4	SynchroniCity cities	Number of RZs with a NGSI interface exposing data sets according SynchroniCity data models	7

Table 27. KPIs global values

#	KPI	Porto	Santander	Antwerp	Carouge	Eindhoven	Helsinki	Manchester	Milan
1	SynchroniCity data sets	7	14	1	0	1	1	4	0
3	Entities	3294	5045	107	0	35	255	55	0

Table 28. KPIs values per RZ

REFERENCES

1. SynchroniCity Consortium, «D4.1 – Validation Methodology Description», 2017.
2. SynchroniCity Consortium, «D2.2 – Data Models Guidelines», 2017.
3. SynchroniCity Consortium, «D2.1.1 – Reference Architecture for IoT Enabled Smart Cities (V1) », 2017.
4. OMA Next Generation Services Interface for Context Management V1.0 [Online]. Avaliable: http://www.openmobilealliance.org/release/NGSI/V1_0-20120529-A/OMA-TS-NGSI_Context_Management-V1_0-20120529-A.pdf
5. FIWARE Data Models [Online]. Avaliable: https://www.fiware.org/developers/data-models/
6. SynchroniCity Data models [Online]. Avaliable: https://gitlab.com/synchronicity-iot/synchronicity-data-models
7. SynchroniCity Consortium, «D2.10 – Reference Architecture for IoT Enabled Smart Cities (V2)», 2018.
8. Draft ETSI GS CIM 004 V0.0.11 (2018-02) «Context Information Management (CIM); Application Programming Interface (API)» [Online] .Available: https://docbox.etsi.org/ISG/CIM/Open/ISG_CIM_NGSI-LD_API_Draft_for_public_review.pdf
9. FIWARE NGSI APIv2 Walkthrough [Online]. Available: https://fiware-orion.readthedocs.io/en/master/user/walkthrough_apiv2/index.html
10. SynchroniCity Consortium, Data Storage API - Historical data access [Online]. Available: [https://synchronicityiot.docs.apiary.io/#reference/data-storage-api-historical/get-historical-data]
11. IETF OAuth Working Group, «The OAuth 2.0 Authorization Framework, RFC6749» [Online]. Available: https://tools.ietf.org/html/rfc6749
12. SynchroniCity Consortium, Security API [Online]. Available: https://synchronicityiot.docs.apiary.io/#reference/security-api
13. SynchroniCity Consortium, SynchroniCity Data Models Validator (SC_DMV) and Reference Zone's Instance Validator tool [Online]. Available: https://gitlab.com/synchronicity-iot/rz-instance-validator
14. FIWARE Sort Term Historic (STH) Comet component [Online]. Available: https://fiware-sth-comet.readthedocs.io/en/latest/
15. FIWARE Identity Manager (IdM) KeyRock enabler [Online]. Available: [https://catalogue-server.fiware.org/enablers/identity-management-keyrock]
16. Synchronicity Consortium, «D2.4 – Basic data marketplace enablers», 2018.
17. Synchronicity Consortium, Marketplace API [Online]. Available: https://synchronicityiot.docs.apiary.io/#reference/marketplace-api
18. Synchronicity Consortium, «D2.8 Report on Basic Reference Zones platform deployment and operational plan», 2018.
19. «Data Catalog Vocabulary (DCAT) – revised edition», W3C First Public Working Draft 08 May 2018 [Online]. Available: https://www.w3.org/TR/vocab-dcat-2/
20. Digitransit Journey planner. [Online]. Available: https://digitransit.fi/en/
21. WSO2 Inc. Identity Server [Online]. Available: https://wso2.com/identity-and-access-management