



WP2 Programmable Smart City

D2.6 Programmable smart city – observations and lessons.

Grant Agreement N°723139
NICT management number: 18301

BIGCLOUT

*Big data meeting Cloud and IoT
for empowering the citizen ClouT in smart cities*

H2020-EUJ-2016 EU-Japan Joint Call

EU Editor: **LANC**

JP Editor: **KEIO**

Nature: Report

Dissemination: PU

Contractual delivery date: 2019-01-01 (M30)

Submission Date: 2019-03-20 (M33)



Co-funded by the EU H2020 GA. 723139 and NICT GA. 18301

ABSTRACT

The BigClouT project has developed a Smart City architecture focused on gathering and exploiting city wide big data that can be harnessed to improve the lives of citizens – enhancing their ‘clout’ in the day to day running of the city. The overall architecture has been implemented and deployed in a number of Smart City trials. This deliverable reports on the experiences using the BigClouT platform instantiations and associated core components as they are deployed in aid of smart city programmability.

City pilots and trials have taken place at all of the four core cities in BigClouT with each city using a specific instance of the BigClouT platform and a number of components, picked to meet the needs of their trial. Experiences from these initial pilots and trials, as well as a number of internal demonstrators are presented.

Disclaimer

This document has been produced in the context of the BigClouT Project which is jointly funded by the European Commission (grant agreement n° 723139) and NICT from Japan (management number 18301). All information provided in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability. This document contains material, which is the copyright of certain BigClouT partners, and may not be reproduced or copied without permission. All BigClouT consortium partners have agreed to the full publication of this document. The commercial use of any information contained in this document may require a license from the owner of that information.

For the avoidance of all doubts, the European Commission and NICT have no liability in respect of this document, which is merely representing the view of the project consortium. This document is subject to change without notice.

Revision history

Revision	Date	Description	Author (Organisation)
V0.1	2019.01.09	Initial ToC	RJL (LANC)
V0.2	2019.01.14	Updated ToC, Added section text	RJL (LANC)
V0.3	2019.01.22	Added Keio text	TY (KEIO)
V0.4	2019.01.24	Added NII text	FC (NII)
V0.5	2019.01.28	Intro, summary etc	RJL (LANC)
V0.6	2019.01.29	OneM2M section added	HM (NTT)
V0.7	2019.02.02	ICCS contribution	OV (ICCS)
V0.8	2019.02.07	ENG contribution	GC (ENG)
V0.9	2019.02.27	Review and CEA contribution	LG (CEA)
V1.0	2019.03.05	Final version	LG (CEA)
V1.0	2019.03.20	Submission	LG (CEA)

TABLE OF CONTENT

1	INTRODUCTION	5
2	OVERALL BIGCLOUD ARCHITECTURE AND TOOLS	5
2.1	BIGCLOUD SYSTEM ARCHITECTURE	5
3	PROGRAMMABILITY	7
3.1	SENSINACT	7
3.1.1	Overview.....	7
3.1.2	Trials/pilots using tools	8
3.1.3	Experiences.....	10
3.2	DISTRIBUTED NODE-RED.....	11
3.2.1	Overview.....	11
3.2.2	Service composition: D-NR studio (visual programming tool).....	11
3.2.3	Edge processing: D-NR platform (fog/edge computing capability).....	11
3.2.4	Constraints: Supporting More Complex, Larger Scale Applications	13
3.2.5	Trials/pilots using D-NR.....	13
3.2.6	Experiences.....	14
3.3	NR-SOxFIRE AND NR-MACHINE LEARNING INTEGRATION.....	15
3.3.1	Overview.....	15
3.3.2	SOxFire node	16
3.3.3	TensorFlow node	16
3.3.4	Trials/pilots using tools	17
3.4	CITYFLOW	17
3.4.1	Overview.....	17
3.4.2	Spatial-Temporal Distributed Edge Environment.....	18
3.4.3	The Problems for Machine Learning Applications in the City.....	19
3.4.4	CityFlow.....	21
3.4.5	Trials/pilots using tools	23
3.4.6	Experiences.....	23
3.5	DATA ANALYSIS COMPONENTS.....	24
3.5.1	System component - recommendation engine.....	24
3.5.2	System component - KNOWAGE.....	27
3.5.3	System component - adaptability framework.....	28
3.5.4	System component - oneM2M Service Layer API.....	31
3.5.5	System component - CDMI Edge Storage.....	36
4	DISCUSSION & SUMMARY	39



LIST OF FIGURES

Figure 1: BigClouT architecture.....	6
Figure 2: Overview of sensiNact Platform and Studio.....	8
Figure 3 : sensiNact in the Grenoble - Inovallée Trial	9
Figure 4: sensiNact resource model applied to the mobility and restaurant information model.....	10
Figure 5: Screenshots from the MyIno App providing information about mobility, restaurants and events.....	10
Figure 6: Visual programming using a drag and drop metaphor.....	11
Figure 7: Distributed deployment to edge devices using D-NR in BigClouT.....	12
Figure 8: Specifying constraints associated with processing modules in a dataflow program	13
Figure 9: Custom nodes developed for CityFlow.....	17
Figure 10: Life Cycle of Machine Learning Applications.....	18
Figure 11: Influence on accuracy due to a spatial-temporal covariate shift	21
Figure 12: Overview of CityFlow	21
Figure 13: Road damage detection application developed by cityflow.....	23
Figure 14: KNOWAGE - Road Infrastructure Management Dashboard.....	27
Figure 15 Self-adaptation for service composition in sensiNact.....	29
Figure 16 Example of an application's ECA rules in sensiNact studio	29
Figure 17 Example of the translated semantic design model.....	30
Figure 18 Application conflicts identified dialog for developers in sensiNact studio.....	31
Figure 19 Application locations in oneM2M System	32
Figure 20 Communication METHOD Portability by API and CLIENT Library	32
Figure 21 Deployment type of AE and CSE.....	33
Figure 22 Application 1 I Need Hurry App.....	34
Figure 23 Application 2 integration with SoxFire through D-NR.....	35
Figure 24 Application 3 Opportunistlc sensing in Fujisawa city	35
Figure 25 Fog Storage System	36
Figure 26 Details of the CDMI Cloud Storage	37
Figure 27 Edge storage details	37



1 INTRODUCTION

This deliverable describes the experiences of the project partners using the BigClouT architecture and tools. It focuses on the use of the tools in the trials and the partner experience using the BigClouT system and tools for trial development.

The BigClouT project has developed a Smart City architecture focused on gathering and exploiting city wide big data that can be harnessed to improve the lives of citizens – enhancing their ‘clout’ in the day-to-day running of the city. The overall architecture has been implemented and deployed in a number of Smart City trials. City pilots and trials have taken place at all of the four core cities in BigClouT with each city using a specific instance of the BigClouT platform and a number of components, picked to meet the needs of their trial. Experiences from these initial pilots and trials, as well as a number of internal demonstrators are presented.

The overall goal of the BigClouT project has been to design a flexible smart city architecture, allowing individual cities to pick and choose the appropriate components for their specific needs. Related to this is a desire to ensure that whatever combination of architectural components are chosen, programming smart city applications on that set of components is easy and as intuitive as possible.

In this report, we provide an outline of the overall architecture of BigClouT and discuss its flexibility. We then introduce the core instantiations of BigClouT, the sensiNact platform used in a number of trials including trials in Grenoble and the SoxFire/Distributed Node Red platform used in Fujisawa. These two instantiations have integration points which are highlighted. In addition to the core platforms, BigClouT has also developed a set of system components that implement core functionality of the architecture but are available as standalone components that can be used with either platform as part of a city trial. The deliverable also introduces these core components.

In all cases, we report on the platform and component usage, and attempt to derive lessons and experiences from using the BigClouT technologies. The aim of the report is to reflect on those experiences and feed into the final 6 months of the project to improve the final trial results.

Note this deliverable was slightly delayed allowing initial trials to finish and report on experiences.

2 OVERALL BIGCLOUT ARCHITECTURE AND TOOLS

This section provides an overview of BigClouT architecture and summarizes its main concepts and element.

BigClouT architecture has been firstly introduced and described in document "D.1.3 First BigClouT Architecture" and its final version has been presented in "D1.4 Updated use cases, requirements and architecture". The BigClouT architecture extends the architecture defined in ClouT project adding new functionalities and capabilities, such as Big Data analysis, Edge computing, etc.

2.1 BigClouT system architecture

BigClouT architecture, depicted in Figure 1, is composed of three main layers:

- ClaaS (City Infrastructure as a Service): which allows to access and perform actions on City Entities;
- CPaaS (City Platform as a Service): which provides functionalities to compose services and applications and to access, analyse and visualize data;
- CSaaS (City Software as a Service): which contains the applications made on top of functionalities exposed by the two previous layers.

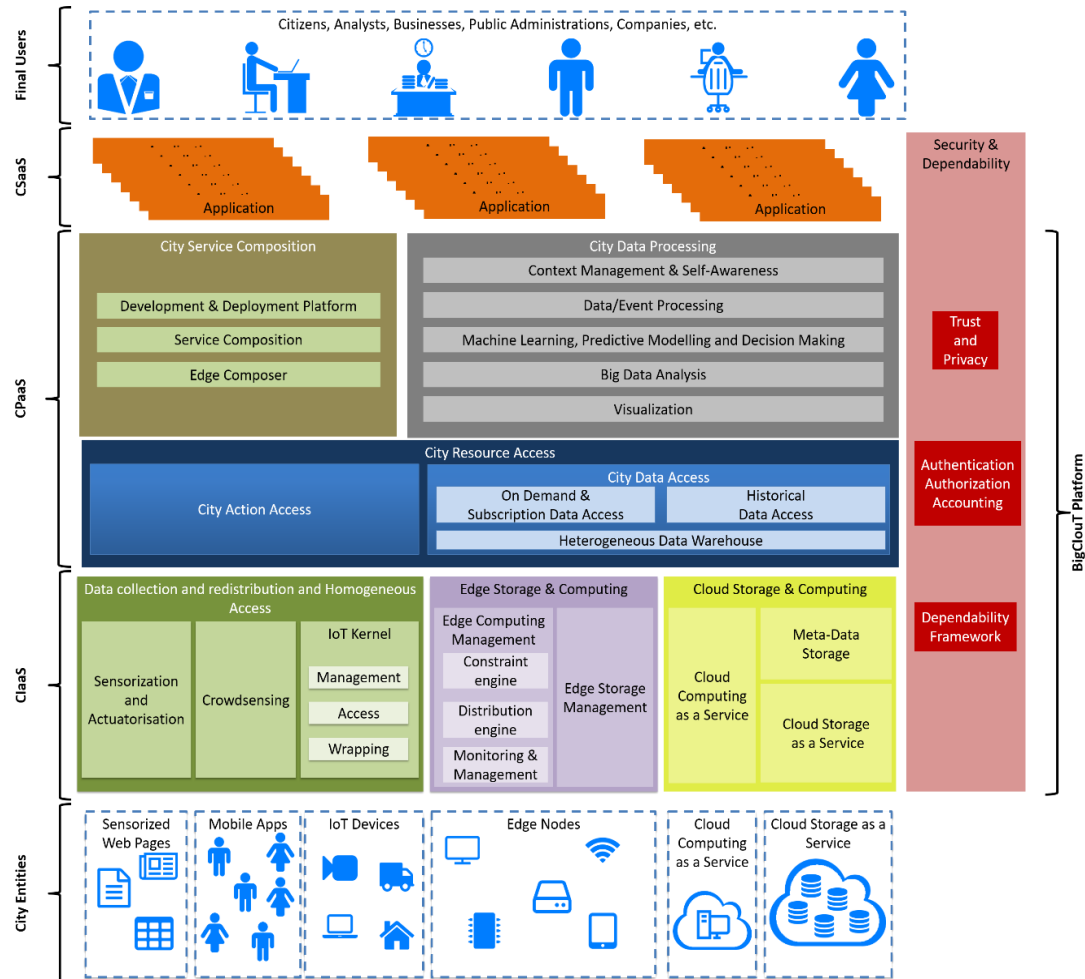


FIGURE 1: BIGCLOUT ARCHITECTURE

Along with the three main layers, the City Entities layers is reported in the architecture. This layer is not strictly part of the architecture and it contains the data sources used to exploit BigClout's functionalities. Moreover, BigClout Security and Dependability layer is reported, this layer is cross to the ClaaS, CPaaS and CSaaS and its main role is to deal with security aspects of the platform.

The BigClout main functionalities are provided by the two layers ClaaS and CPaaS. These two layers are composed by several modules and submodules. In particular, ClaaS layer is composed by the following modules:

- **Data Collection and Redistribution and Homogeneous access:** which collects city data, provides the access to it and manages the interactions with actuators. This module exposes these functionalities thanks to its submodules: Sensorisation and Actuatisation; Crowdsensing; and IoT Kernel.

- **Edge Storage & Computing:** which manages edge storage and processing capabilities thanks to its submodules: Edge Computing Management and Edge Storage Management.
- **Cloud Storage & Computing:** which manages cloud computational and storage resources allowing to interact with them thanks to its submodules: Cloud Computing Connector, Cloud Storage Connector and Meta-Data Storage.

CPaaS layer is composed by the following modules:

- **City Resource Access:** which provides access to some of the functionalities exposed by the CIaaS layer (for instance, it allows to access historical data and (near) real-time data and to perform action on City Entities) thanks to its submodules: City Action Access and City Data Access.
- **City Service Composition:** which allows to develop and deploy application on top of CIaaS layer thanks to its submodules: Development & Deployment Platform, Service Composition and Edge Composer.
- **City Data Processing:** which provides capabilities to perform data/event processing, big data analysis and visualization, context management thanks to its submodules: Big Data Analysis; Visualization; Machine Learning, Predictive Modelling and Decision Making; Data/Event Processing; Context Management & Self-Awareness.

3 PROGRAMMABILITY

BigClouT has developed a number of tools to enhance programmability, these include the tools developed for the sensiNact instance of the BigClouT architecture, the Node Red based BigClouT visual programming tool, D-NR and the tools and libraries developed for the SoxFire instantiation of the BigClouT architecture. This section discusses these technologies separately and describes their usage and experiences from application development for the core trials and for a set of small-scale demos.

3.1 sensiNact

3.1.1 *Overview*

The Eclipse sensiNact consists of a software platform enabling the collection, processing and redistribution of any data relevant to improving the quality of life of urban citizens, programming interfaces allowing different modes of access to data (on-demand, periodic, historic, etc.) and application development and deployment to easily and rapidly build innovative applications on top of the platform.

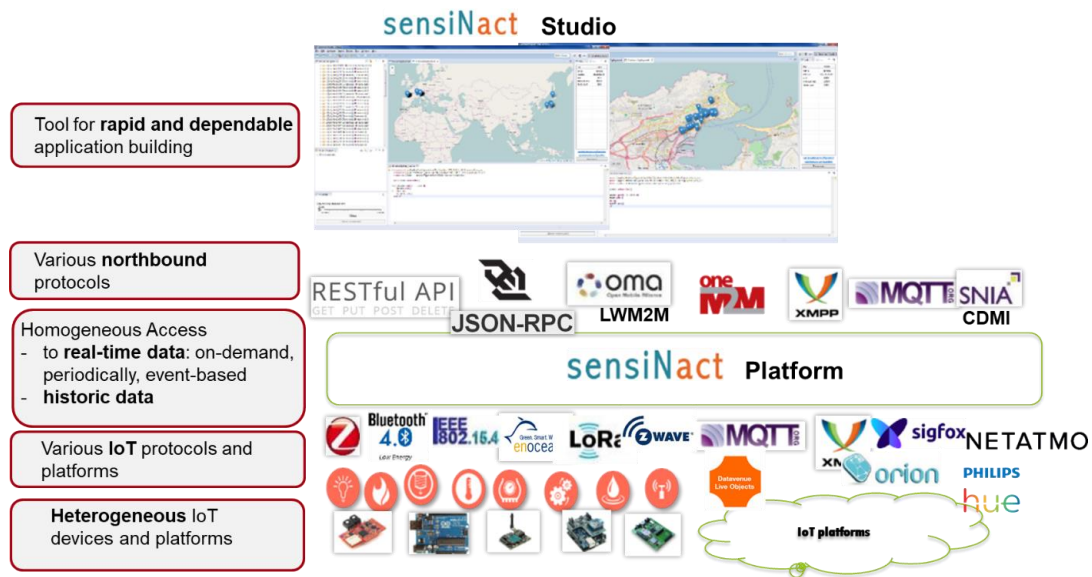


FIGURE 2: OVERVIEW OF SENSINACT PLATFORM AND STUDIO

At the heart of sensiNact lies its service-oriented approach in which IoT devices expose their functionalities in terms of services (temperature service, presence detection service, air quality monitoring service, alarm service, etc.). Each service then exposes one or several resources such as sensor data or actions. Building applications thus become a matter of composing sensing services with actuation services. Loosely coupling between the devices and the services they implement makes the composition of services more dynamic and adaptable to the changing context, not only in the software environment (increasing CPU or memory usage, low battery, reducing quality of measures, etc.) but also in the physical environment (replacing sensors, changing localization, etc.).

3.1.2 Trials/pilots using tools

The main trial that used the sensiNact platform is the Grenoble trial. Located in Grenoble, CEA-LETI has been the main partner of the Grenoble pilot deployment.

The use case identified by the Greater Grenoble City Area is about increasing the quality of experience of employees in an industrial zone located in the Grenoble Greater Area, namely innovallée¹. The objective is to provide to the employees working in the zone useful and practical real-time information about the transportation, restaurant options, cultural or sport events taking place in the zone, etc. Besides, the information about the employees, their interests, localisation, preferred mobility modes, the usage of the mobile application provides useful information for the Innovallée association that is in charge of managing the zone.

sensiNact gathers following data from various data sources:

- Mobility related open data from Grenoble Metropole
<https://www.metromobilite.fr/pages/opendata/OpenDataApi.html>
- Web service from the company Elior which is managing several restaurants in the zone. Other restaurants will be later integrated to the app.

¹ <https://www.inovallee.com/>

- Event information from the Inovallée database. A new event management mechanism will be integrated soon. This new mechanism includes possibility for the users to create and publish events to the system.

- Information about the companies located in the zone.

- Other information that can be added in the future.

In addition to the data gathered from those data sources, data generated by the users (usage of the mobile application, profile information, location information, events created by the users, etc.) will be collected by sensiNact in the 2nd version of the application.

The important added value of the sensiNact is that all these information are available in one common place, with common APIs in a common data model, which facilitates the collection and analysis of the data. BigClouT tools analyse the data and provide personalised recommendations to the application users based on the context data they provide (localisation, user profiles, usage of different parts of the application, etc.). The Figure below illustrates the architecture of the integration work.



FIGURE 3 : SENSINACT IN THE GRENOBLE - INOVALLÉE TRIAL

3.1.3 Experiences

sensiNact has been already used in many IoT and smart city applications integrating various IoT protocols and data platforms. In this trial, we had the opportunity to use sensiNact in a very new domain: monitoring of industrial zones. This allowed us to validate the adaptability and extensibility of sensiNact in such new domain. This trial allowed us integration of at least 3 new data sources to sensiNact and apply our service/resource model to those new data sources.

For instance the sensiNact model, which is based on three layer service provider -> service -> resource approach, has been applied in modeling of restaurants and mobility cases as illustrated in the Figure below.

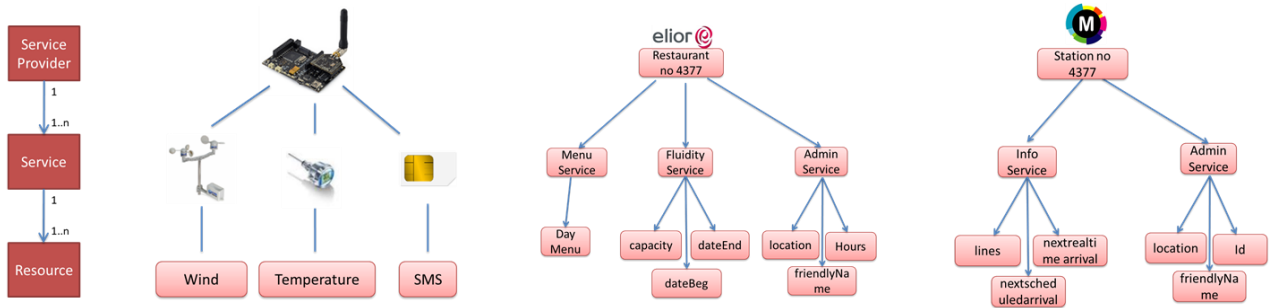


Figure 4: sensiNact resource model applied to the mobility and restaurant information model

Thanks to the integration experience in the context of this trial, we could validate one of the main features the BigClouT data collection platform (aka sensiNact): easy and quick integration of new data sources.

The following screenshots from the application shows the information gathered by the MyIno App from sensiNact.

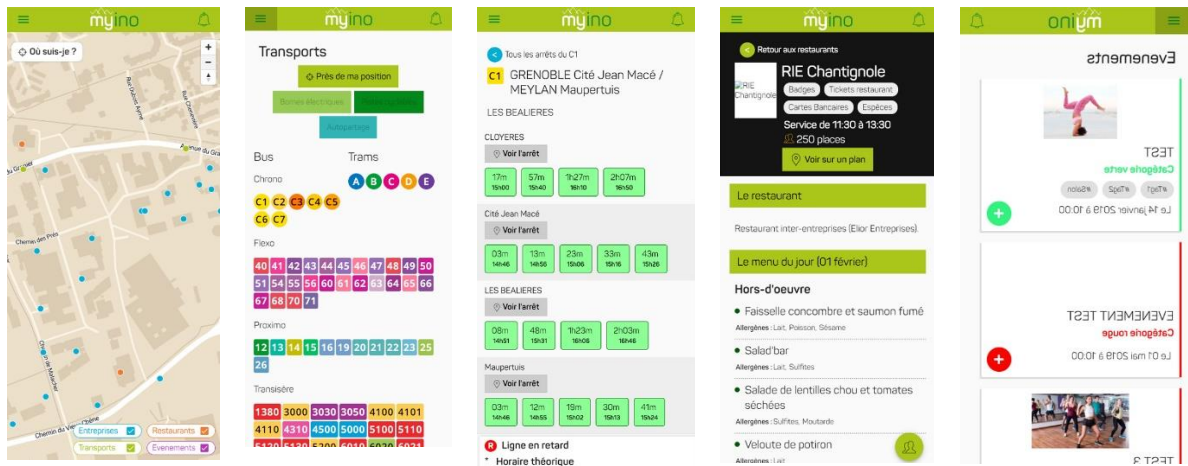


FIGURE 5: SCREENSHOTS FROM THE MYINO APP PROVIDING INFORMATION ABOUT MOBILITY, RESTAURANTS AND EVENTS

Further analysis and evaluation on performance of the system and user satisfaction will be performed in the coming months during and after the execution of the trial.

3.2 Distributed Node-RED

3.2.1 Overview

SoxFire, the core data distribution developed by Keio university has been combined with the Distributed Node-RED (D-NR) programming framework developed by Lancaster University and used in a number of demonstrators and trials.

Soxfire uses an extension of the XMPP standard to provide a robust and efficient data communication framework. It enables physical and virtual sensors to easily distribute data to core BigClouT architecture components and provides the base level of the architecture in a number of the BigClouT trials

The distributed data flow programming tool D-NR is split into two core components: Studio and Platform. Studio provides a visual programming tool allowing service composition by dragging and dropping application components onto a visual drawing board and wiring the components together to form a flow. While the D-NR platform supports edge processing and dynamic load balancing across the BigClouT network.

3.2.2 Service composition: D-NR studio (visual programming tool)

Service composition uses a set of pre-defined building blocks that are wired together to form the application logic.

In a basic air quality application is shown that reads data from the BigClouT data warehouse (sensor data from Bristol) and visualises the data on a dashboard.

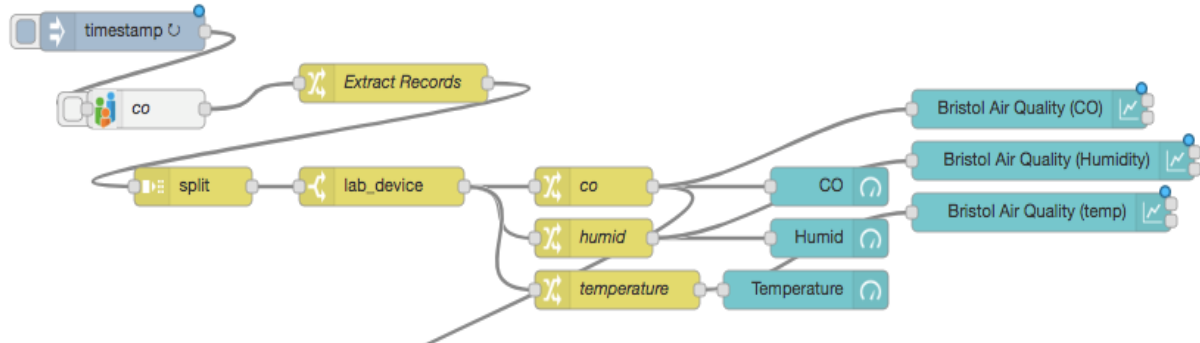


FIGURE 6: VISUAL PROGRAMMING USING A DRAG AND DROP METAPHOR

The data flow from left to right through the application, traversing the 'wires' between the processing nodes. Each node receives data, carries out some processing and sends the data to the next processing node in the flow.

3.2.3 Edge processing: D-NR platform (fog/edge computing capability)

To extend Node-RED to meet the BigClouT need for a service composition tool with support for edge processing we addressed a number of key issues:

- Developing a model to describe devices and their capabilities and incorporating into the Node-RED development tool.

- Supporting a language transparent mechanism to allow nodes in the application flow to be moved to remote devices.
- Defining a constraint model that allowed application developers to specify constraints for different parts of their application flow, which then drove the underlying distribution and replication mechanisms.

We introduced the notion of *device* to the dataflow language. Accordingly, every node in a dataflow program is augmented with a new *device Id* constraint that specifies on which device the node should be deployed and run. For example, a node can be constrained to be deployed on an edge device, a mobile host, a cloud server or on any intermediary device across the edge to the cloud.

The second augmentation made to Node-RED was the notion of "remote wires" or "remote arcs". Since the nodes may run on separate devices, the existing Node-RED wires - which represent dataflow links between processing nodes - have to support inter-device communication to handle the situation where a flow is broken up and its nodes are distributed to several devices. This is implemented using a publish/subscribe communication mechanism that binds the nodes together. The key idea is to leverage the node identifications as the topic for publishing and subscribing. Further, a flow transformation process is applied so that the nodes that do not meet the deployment requirement (e.g. run on "mobile", "laptop" or "server") will be replaced with a *wire in* or a *wire out* node. *wire in* nodes subscribe to the communication broker so that it can receive data from the external node running on a different device. *wire out* nodes receive data from the local node and publishes it to the communication broker so that the *wire in* node from the other side can pick it up. Figure 7 illustrates this process of supporting the distributed deployment of a Node-RED flow across multiple devices.

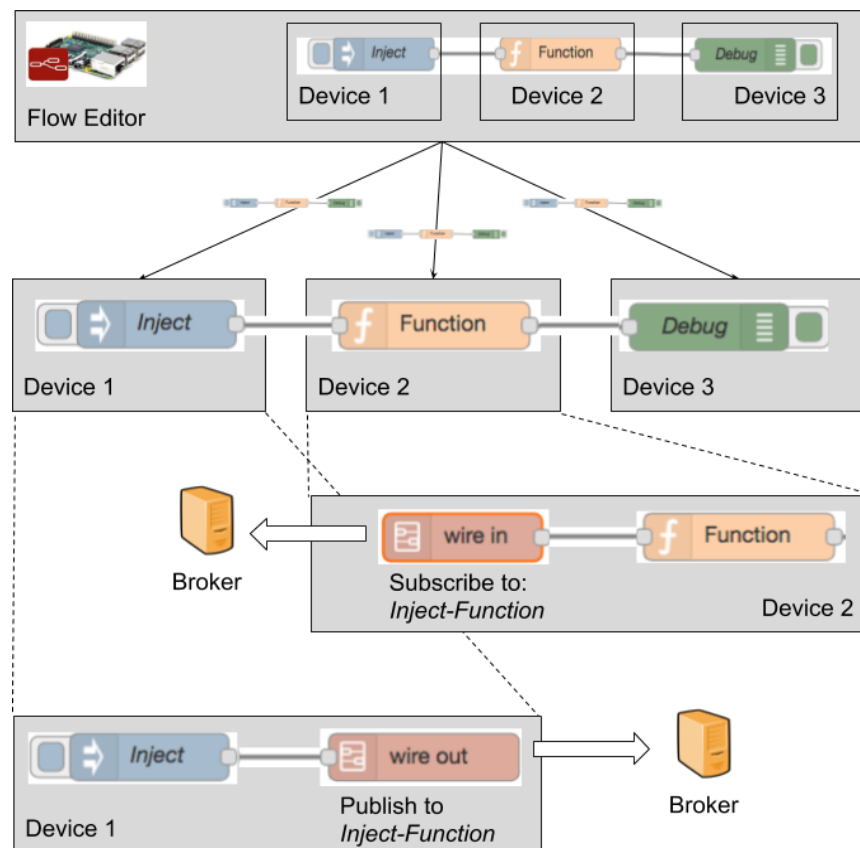


FIGURE 7: DISTRIBUTED DEPLOYMENT TO EDGE DEVICES USING D-NR IN BIGCLOUD

3.2.4 Constraints: Supporting More Complex, Larger Scale Applications

To support larger scale BigClouT applications, we augmented the basic device specification capability with a constraint tool. This allowed developers to define a set of constraints for a part of the flow, e.g. a node should run on a device with 4MB of memory, a 4 core CPU and is physically located in the Henleaze area of Bristol city.

This more sophisticated mechanism allows scenarios such as:

- A sensor node mounted on a moving vehicle could be restricted to operate in a certain location.
- A vision processing node might be restricted to operate in a more capable computing device.

To address these needs, we introduced the *constraint* primitive as a broader abstraction that specifies how a node is deployed and run in a distributed computing setting. Accordingly, every node in a dataflow program is augmented with a *constraint* property that defines how the deployment is carried out. In BigClouT, a *constraint* involves the requirements on device identification, computing resources such as CPU and memory and physical location.

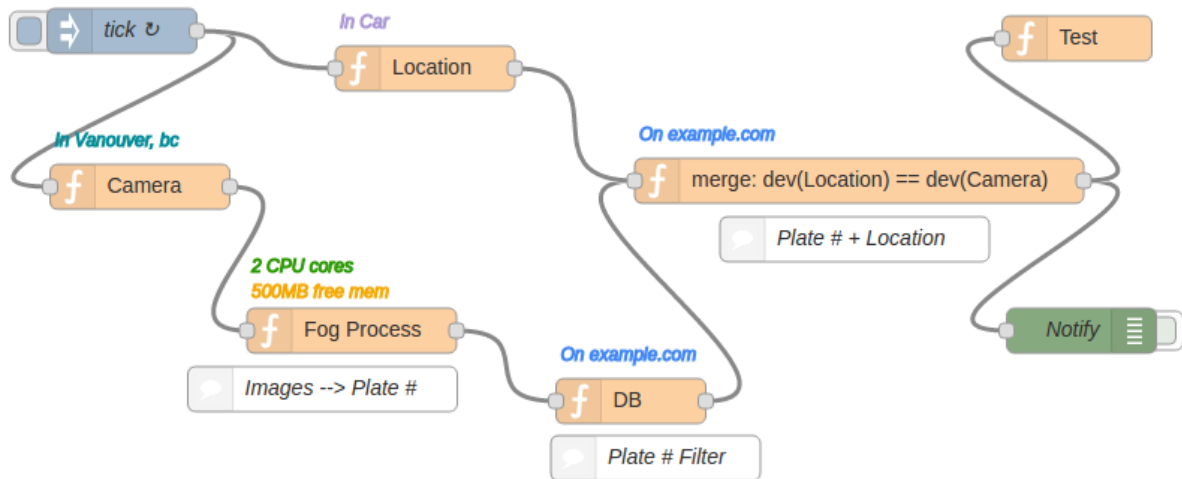


FIGURE 8: SPECIFYING CONSTRAINTS ASSOCIATED WITH PROCESSING MODULES IN A DATAFLOW PROGRAM

The goal is to make the application model more suitable for a class of fog-based applications that are heavily dependent on the context associated with the edge devices they operate on. As a result, the developer can not only specify which type of device a node should run on (e.g. mobile, server or laptop, etc.) but can further constrain where the node should run based on a variety of aspects such as memory size, processing capability, location etc. To address this need, we added support to allow application developers to specify these node constraints via the programming user interface. As can be seen in Figure 8 (above) a developer has associated a set of constraints, e.g. on server named **example.com**, or any device with **2CPU cores** and **500MB of free memory**.

3.2.5 Trials/pilots using D-NR

D-NR has primarily been used in the Fujisawa trials with a focus on infrastructure monitoring. In initial versions of the trial, D-NR was primarily used as a co-ordination tool to deploy and manage replicated code on a set of garbage collection trucks. However, in later versions of the trial, D-NR

was augmented with machine learning capabilities (described below in 3.3) and ultimately evolved into CityFlow, a complete framework for developing flow-based city applications that leverage AI technologies (see section 3.4).

3.2.5.1 Road infrastructure monitoring – basic scenario

In Japan, the problem of aging road infrastructure is increasingly gaining importance since around half of the road network (including bridges) was originally built 40 years ago. To repair roads and road-related equipment such as mirrors and road markings, it is important to understand which area of the city has what kind/level of problem. Currently, the municipality just inspect road condition of limited areas (mainly national roads) once every several years. Thus, the condition of most roads in the city is not monitored.

During discussions with the road management section in Fujisawa city, we found that they need to specify priority for roads to be repaired. In addition, it is desirable to check actual road status visually to confirm the necessity of repair. To meet the requirements, we provided the following scenario which contains three phases of road infrastructure management. The scenario leverages garbage trucks as sensors, something that was demonstrated at the first review of the project. The full details of this trial are reported in the sections 3.3 and 3.4, so only a brief overview is provided here.

- **Phase1 Macro Sensing**

Garbage trucks cover more than 98% of the road network in a week, so we can collect complete/full road health condition by attaching sensors and edge computational resources to garbage trucks. Uploading all of the road images is unrealistic because of limited network bandwidth. Thus, we analyse road status such as condition level with GPS coordination at edge-side (computer on garbage truck). The analysis leverages an edge analysis component called DeepOnEdge developed as part of WP3. Analysis results from all garbage trucks are collected and published to the BigClouT data warehouse by using the distributed data flow component called distributed Node-RED (D-NR) developed in WP2.

- **Phase2 Priority Decision**

Once the road condition level of the whole city is determined, the priority for each road is determined, i.e. which roads must be repaired as soon as possible. For setting reasonable priority, in addition to road damage information, various conditions such as how many people/cars use the road should be considered. For combining and analysing different data, we utilise the big data analysis component called KNOWAGE developed in WP3. According to defined priority, micro sensing operation is sent to each garbage trucks by D-NR.

- **Phase3 Micro Sensing with Privacy Protection**

To meet the requirement that city officers see the actual road image (i.e. a visual confirmation), the last phase is to upload actual image of roads to be repaired from garbage truck sensors. Image taken by garbage trucks may contain privacy information such as faces of pedestrians or car numbers. Therefore, such privacy information should be removed before sending the image. Which means that “anonymisation” of the images must take place by using the DeepOnEdge component. Finally, anonymised images from specific roads are collected through D-NR, and city officers can make plans of road repairs.

3.2.6 Experiences

As mentioned above, the details of the use of D-NR in the Fujisawa trials is reported in the next two sections. As such, we will leave the main experiences/lessons for those sections and simply report here on experiences using D-NR in internal demonstrators.

The D-NR framework has also been used in Lancaster to implement a series of small-scale tests that are designed to align with trials carried out in Bristol.

Key lessons learnt are:

- A Visual programming tool is powerful for prototyping, for small scale application programming and excels at a rapid development of flow-based data centric applications. We have used it in a number of ways, but its ease of use has meant that it has been used repeatedly to:
 - provide a glue framework to tie components together
 - develop small data applications that access smart city data, quickly process and visualise before storing in long term storage
- Constraints were able to support the test scenarios and offer a powerful way to drive distribution and edge processing. Deciding on distribution of application components and managing real time edge processing in a dynamic system is a complex task. Especially as the application scales. The use of a visual tool to attach constraints to the application components and work in an intuitive manner when expressing constraints was reported to greatly ease the programming task
- Large scale applications often require external programming support, e.g. call out to external software. While it was hoped, at the initial stage of the project, that the D-NR tool would be expended with a number of built in components that implemented sophisticated logic, one of the key learnings from our experiences is that it is often easier to use external components developed outside D-NR. We saw this in particular when attempting to exploit sophisticated AI algorithms. While possible to develop within D-NR it was quicker and more efficient to use existing components and so a mechanism was developed to wrap external components and use them as if part of a native D-NR flow. This is highlighted in the sections below when discussing Machine learning and Tensor Flow.

3.3 NR-SoxFire and NR-Machine Learning Integration

3.3.1 *Overview*

As we outlined in previous deliverables, SOXFire is one of main component for urban sensor data distribution tool in BigClouT architecture, especially for federated multi-community purpose. In addition, SOXFire connects several components which collects urban data from website, SNS, IoT garbage trucks, or people (MinaRepo). To access urban data from SOXFire in city programmable tool - D-NR, we developed SOXFire node for Node-RED.

In addition to access urban sensor data from SOXFire or different data resources, programmable tool should provide easy-to-use ML(Machine Learning) tool for understanding complex city context. To classify city context from various kinds of city data, adapting ML algorithm should be one of effective ways. However, current ML requires deep knowledge and experiences - it is still far away to use ML algorithm in programmers of city staffs or novice users. Thus, we provide ML node for adapting and executing ML process in Node-RED environment. In our prototype system, we leverage TensorFlow component since TensorFlow is one of the major ML tools which are widely used in data analysis.

3.3.2 SOXFire node

We developed a node implemented in D-NR that handles Sensor-Over-XMLPP (SOX) as shown in Figure 9. SOX is the specification of SOXFire which is a universal sensor data exchange system. This utilizes the Internet protocol of the open XML format (XMPP), typically used for chat communications, to represent the meta information. Accordingly, by using SOX nodes, the data can be treated uniformly from the physical sensors and the virtual sensors.

This feature is useful for the data used in ML techniques in the city. Furthermore, since the ML model is distributed to the edge devices all over a city to conduct prediction locally, SOX can be used.

3.3.3 TensorFlow node

We also implement TensorFlow nodes as depicted in Figure 9. These nodes aim to help in developing an application in which deep learning is used for training and predicting. These nodes are implemented using javascript. Though, of course, the functions of these nodes offering such as 'predict' and 'train', are mainly provided as a low-level library by tensorflow-js², these functions require users to code lines expert knowledge. This is not suitable for the goal of easy city programmable tool that it is supposed that the tool is used by not only the expert of machine/deep learning but also beginners such as local governments or city officers.

In order to achieve the goal of easy city programmable tool, we have to implement the nodes as high-level functions to make deep learning procedure simple and easy. For example, in general, developers need to design a neural network architecture, then determine the hyper-parameters for the network (model) optimization. However, it is almost impossible for beginners to do as well. Therefore, we have to mask those procedure from beginners.

According to this strategy, we wrap functions that need to execute a deep learning approach to six kinds of nodes as follows:

- train Node

This node provides deep learning networks optimization. The hyper-parameters for optimization, such as 'learning rate', 'batch size' and several optimizers are provided.

-predict Node

This node is able to use for predicting. Thanks to the function of tensorflow-js, developers just are required only to prepare the URL which indicates the model architecture and its weights. Indeed, to use this node, developers inputs the URL.

-dataload Node

This node provides the function that loads data as html canvas format so that the data such as images are transformed to array format.

-img2tensor Node

This node is very important node because most of deep neural networks require their input as tensor and this node transform the data from array format to tensor format.

² <https://js.tensorflow.org/>

-classify Node.

This node is implemented for post-processing. This classify node requires the class labels to translate the input vectors to labels, for the input, which is the output of neural networks, represent categorical distribution of classes.

-dataset Node

The dataset node is used for pre-processing the dataset, such as divide the csv files into data and labels, then split them into train dataset and test dataset.

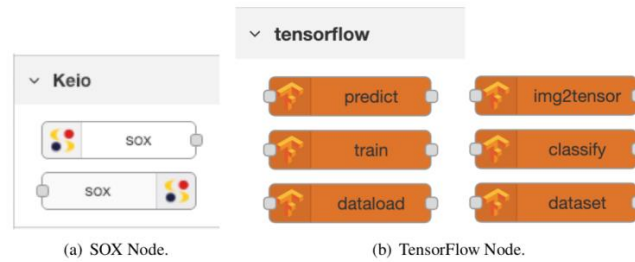


Figure 9: Custom nodes developed for CityFlow.

3.3.4 Trials/pilots using tools

3.3.4.1 Road infrastructure monitoring – basic scenario

This trial is same described in 3.2.5.1. Please refer the section for the detail.

3.3.4.2 Experiences

Because these tools are integrated in CityFlow system with D-NR, we will explain our experiences in Fujisawa in the next section.

3.4 CityFlow

We have designed and implemented CityFlow, which supports the development of ML applications in city which is spatial-temporal distributed edge environment.

3.4.1 Overview

In order to realize smart cities, there has been extensive research into using machine learning (ML) techniques. For example, citizens' behaviour prediction by using GPS trajectories of their smart devices or taxis, and road damage detection from video recorders mounted on automobiles. In both cases, city data is gathered into centralized cloud servers for analysis.

This centralized approach is well suited to ML algorithms, but unfortunately does not reflect the reality of Smart Cities. Typically, cities have a multitude of sensors and 'things' (e.g. automobiles, mobile devices and robots) that are distributed throughout the city and communicate with each other via the Internet (e.g. Wi-Fi, LTE, 3G or Ethernet).

This results in city data exhibiting a wide variation in time and space both in short term, i.e. daily cycles, and in longer term cycles as the city, its infrastructure and its citizens evolve.

In short, the city is a spatial-temporal distributed edge environment (STDEE) and requires we adapt our ML techniques to this environment.

However, developing and operating ML applications in a STDEE is not straightforward. This is primarily because the development process, which is already complicated because of the need for multiple trials to aid learning, needs to also accommodate the distributed nature of the STDEE. Obviously, this leads to significant development costs, both in terms of time and resources, and is by nature inherently complex.

CityFlow makes it possible to easily describe the flow of data which comes from devices in the city, and to deploy trained ML models to a variety of devices distributed throughout the city. By making these processes easy, we can conduct data pre-processing and preservation quickly and repeat proof of concept (PoC) that verify the performance of the ML model.

CityFlow is built using combining Distributed Node-RED (DNR) and customized Node-RED node of SOXFire and TensorFlow as we explained previous section.

3.4.2 Spatial-Temporal Distributed Edge Environment

With the developments in IoT technologies, an enormous amount of data from sensors in the city can be obtained. This covers traditional city infrastructure such as water, electricity and lighting but also increasingly comes from smart devices such as smart transportation, smart homes and offices and perhaps most importantly citizen data from their personal devices. At the same time, an increasing number of small edge devices such as embedded computers being deployed: such as NVIDIA Jetson TX2 (embedded computers), Google Embedded TPU and Xilinx Spartan6 (FPGA).

As the capabilities of these edge devices increases, regardless of their low price, it is possible to use them to support distributed, ML applications. For example, they are already used in domains such as agriculture. Simultaneously, some work has begun to explore deep learning on edge computers for mobile applications. Finally, an important trend is towards treating city information from web service and also the statistics of the city from open data thanks to the spread of IT technology, such as cloud computing technology, which treats the data sources as virtual sensors.

While these trends are leading to significantly more city data being gathered, it is also clear that the spatial-temporal nature of that data is changing. Significant amounts of data are spatial in nature, e.g. where sensors are mounted on garbage trucks, the data is gathered from a variety of locations as the truck travels through the city. Equally, data is gathered during the working day, but not at night - hence it exhibits a high degree of temporal variance. This variety is common to many sources of data, for example data from citizens' mobile devices. Consequently, cities where the IoT and the edge computing technologies are spread throughout exhibit key characteristics of STDEEs.

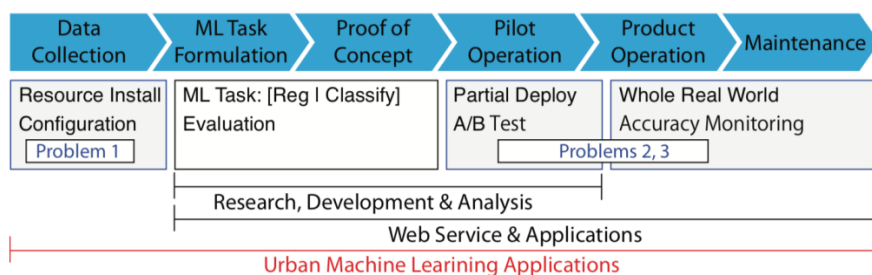


Figure 10: Life Cycle of Machine Learning Applications

In general, the phases (or life cycle) which are conducted in ML application development are shown in Figure 10. In the first phase, in order to tackle a particular city problem, data analysis such as statistical estimation is carried out for understanding the gist of the problem and formulate it as a ML task. Most of the task formulation becomes either regression tasks or classification tasks. Simultaneously, the ML approach would be determined by whether the data is labelled (or annotated) fully, partly, or not at all: supervised learning, semi-supervised learning, unsupervised learning.

In the second phase, to perform the formulated ML task, we design and train the ML model using a given dataset. Then, we experiment to evaluate the model performance as regards its accuracy. In this paper, we refer to this process as the Proof of Concept (PoC). If the model performance is poor, it may be necessary to start over by re-formulating the problem.

As a result of the PoC, if the ML model is able to perform adequately, then we implement and install this ML model into the edge environment and verify whether the model is effective in the real world (we refer to this as the pilot operation). During this phase we adjust for accuracy; identifying factors such as data quality or quantity, which can be improved or factors such as the capacity of the model which may result in re-design of the model.

Finally, we shift to the product operation phase during which, in order to improve the model, we periodically re-build the dataset with additional new data which continues through to the maintenance phase.

In summary, in the development and operation of the ML application, we repeat the exploratory phases task formulation, PoC, pilot deployment, production operation, and maintenance.

3.4.3 The Problems for Machine Learning Applications in the City

While urban ML applications, which are offered to solve the urban problems (e.g. parking availability monitoring are proposed, there is some difficulty to deploy to the real world. The data used by those applications are collected by sensors which are installed temporarily, with limited coverage, and in a precise controlled manner. Furthermore, they have to select the sensor devices which satisfy their application requirements. We call these technical concerns spatial-temporal device-data dependency, which occurs in the phase surrounded by blue line in Figure 10.

While urban ML applications follow this lifecycle, there are problems in the phases that are surrounded by blue line in Figure 10, respectively.

Problem 1: Urban Data Collection and Pre-processing

In statistical machine learning, it is generally assumed that the data have already been collected, formatted and normalized to use for the training dataset - often using offline processing and assuming homogenous datasets.

In contrast, cities exhibit highly heterogeneous datasets, which change over time as the city changes and where data is streamed in real-time. This results in increased processing and storage and requires a dynamic learning model that retrains as data changes.

While there exist a number of algorithms which are able to handle streaming data, they often assume cloud-based servers with load balancing and scaling, and work on homogenous datasets. This is often not the case with real world city datasets.

In addition, we usually conduct some pre-(post-)processing of the data. For example, for privacy protection, since the data from edge devices is often bound to the real world, there is a risk of

privacy invasion. For example, camera data from city sensors, or car data captured in real time has significant privacy issues. However, the diversity of devices makes it hard to cope with the data processing required for dealing with this privacy invasion.

Problem 2: The Machine Learning Model Execution Environment.

After the PoC phase, it is necessary to distribute the trained ML model to all (or a subset) of the edge devices in the city. In typical ML applications, high-performance cloud computers are utilized. This is often necessary because the ML model has a large number of parameters requiring significant memory and high-performance CPUs. However, it is often too expensive for cities to own and manage such servers due to budget constraints. In contrast, we can use edge devices in a STDEE city that typically have significantly lower specifications than cloud-based servers. However, while we can use traditional approaches on edge devices with reasonable computational resources, the latest approaches which have good performance, such as deep neural networks, are not always appropriate due to their resource needs. In order to benefit from their performance, we need to divide the ML model into subsets of small ML model or processes that can be distributed across a set of edge devices or compress it to load it on the edge device memory. Additionally, since the edge devices are highly heterogeneous, it is not just a simple matter of distributing partial modes to a set of edge devices, rather the constraints and context of each device needs to be considered as part of any distribution algorithm.

Problem 3: Covariate Shift of the City Data

In statistical machine learning, typically it is required that there is no difference between the marginal probability distribution of the training dataset and that of the test dataset.

Namely, assuming the distribution of training dataset is p and that of test dataset is p' , $p(x) = p'(x)$ is required. In an STDEE based city, the network status and the context of the location where the edge devices are installed, changes. Therefore, the marginal distribution becomes different, although the relationship between the data x and the desired output y , that is posterior $p(y|x)$ is consistent. This phenomenon is known as covariate shift: $p(x) \neq p'(x)$, $p(y|x) = p'(y|x)$. When considering smart cities, we use the hypothesis that the covariate shift exists in the city, as the distribution of the data from the city often varies significantly in spatial and in temporal domains. We call the covariate shift caused in the city as Spatial-Temporal covariate shift. The covariate shift of the city data is illustrated in Figure 11.

The prediction accuracy of the model is high in the shopping street, because the model is trained with the data collected in the shopping street. In contrast, in other areas away from the shopping centre, the more the data distribution differs from that of the shopping street, the lower the prediction accuracy is owing to the ST covariate shift. For instance, supposing car detection based on video image analysis is required, a model which is trained with the video of shopping streets can detect cars with high accuracy in similar streets. By contrast, at the sea coast, the model struggles to precisely detect cars because the background is sea water instead of buildings. Similarly, if the model is trained with the videos taken in daytime, it is difficult for it to detect cars during the night, even though the location is the same. Therefore, it is important to consider these ST covariate shifts when we develop the ML applications for the smart city.

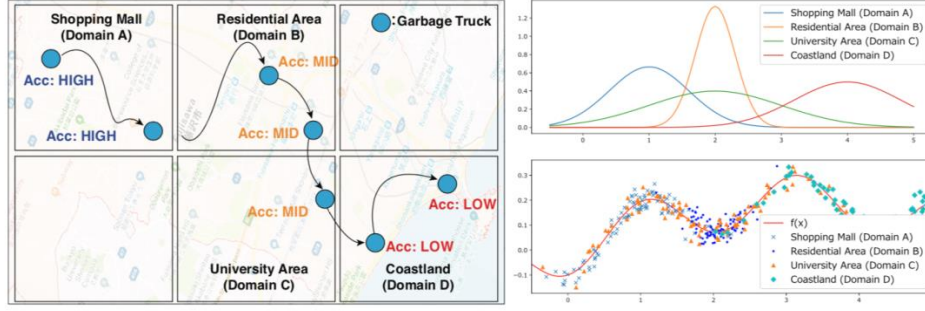


Figure 11: Influence on accuracy due to a spatial-temporal covariate shift

3.4.4 CityFlow

To remedy the spatial-temporal device-data dependency, we developed CityFlow, which is the system combining D-NR and SOX. The overview is depicted in Figure 12.

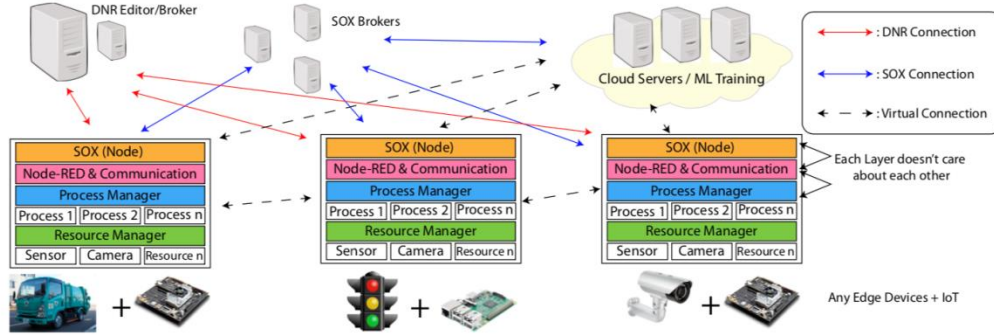


Figure 12: Overview of CityFlow

3.4.4.1 Design Policy

In order to address the issues described previously, we construct an integrated development environment, called CityFlow. This can be used to flexibly realize the life cycle of ML application development in the city. While typical ML applications utilise a centralized processing system, it is difficult to effectively handle the large variety of data sent from the enormous number of edge devices in the city. In contrast, the ML applications, developed within CityFlow, adopt a distributed processing approach, better matched to the spatial-temporal distributed edge environment within a city.

Edge Devices and Pre-processing Virtualization

While the typical ML lifecycle assumes that the data has been already collected into databases, the same does not apply in a city environment. To tackle this, CityFlow collects the data by controlling the edge devices and facilitating the communication between those devices. When collecting the data, it is important to consider the difference between the specification of edge devices and their installed environment. For instance, the data resource could be a virtual sensor such as a web service API or a physical sensor such as an accelerometer. Although the pre-processing should be performed at the sensor where the data is collected, the format of data often differs between platforms, creating additional complexity.

In order to deal with this, CityFlow abstracts away the different edge devices and different data formats to ensure interoperability. Thus, developers are able to concentrate on developing ML models without concern for the underlying technologies and protocols. Furthermore, this abstraction realizes simplified sharing and reusing of these resources among additional developers or applications.

Resource Management and Efficiency Improvement

The task formulation phase and PoC phase in the ML lifecycle is essential to start developing a ML application. In general, it is necessary to select the required data from databases to shift the phase from the task formulation to PoC. However, the presence of data which is not used results in inefficiencies. Therefore, it is efficient for networks and storage mediums to filter the necessary data before storing it into databases.

There are several ways to filter data, including:

Explicit selection: Specify the unique device ID to obtain data from it.

Implicit selection: Specify by spatial and temporal range, such as geolocation or time span etc, and collect only data from devices in that range.

The capability of an edge device, such as CPU and memory, is often lower than that used in cloud computing. Consequently, they cannot perform data pre-processing or model prediction in situations where an edge device receives an enormous amount of data. To effectively handle these situations, the device sends the data to multiple nearby edge devices to balance the load.

Domain Adaptation for Spatial-Temporal Covariate Shift

It is necessary to handle the ST covariate shift to develop ML applications in the city scenario described above. However, there are two challenges:

Firstly, since the statistical machine learning including deep neural networks cannot perform extrapolation, we need to take the approach that treats the extrapolated data: ST covariate shift. For instance, we can adopt domain adaptation learning. They assume that a covariate shift occurred between training data (source domain) and test data (target domain). With this assumption, they provide three types of models: the feature extractor, which outputs the feature from the common probability distribution wherever the input from the source domain or the target domain, the solving model which outputs values for the task (regression or classification), and the discrimination model which determines whether the input comes from the source or the target.

Although adopting these models is one of the efficient solutions for domain adaptation between two domains, the number of domains in the city might be more than two; it is difficult for the city to adopt these models without change. Therefore, to solve the task, CityFlow is required to distribute the models to edge devices in each domain.

The second challenge is the detection of ST covariate shift, which is an estimation of the boundary between the domains as shown in Figure 11. In typical ML applications, the meta-information, such as the device IDs and locations, is attached to the data because of being centralized; this is problematic for treating ST covariate shift. Accordingly, CityFlow is required to provide a function that is capable of monitoring the performance of ST covariate shifting and estimating when the boundary occurs.

3.4.5 Trials/pilots using tools

This trial is the same as described in 3.2.5.1. Please refer the section for the detail of the trial itself.

3.4.6 Experiences

Before actual deployment to Fujisawa road infrastructure monitoring, we designed the NR flow for the trial shown in Fig 13.

The top half of the flow in Fig 13:

In order to deploy the networks to edge devices and cope with ST covariate shift, the neural network is separated into feature extractor node and damage detector node and deployed to different devices on the truck, respectively. Consequently, we can adapt domain adaptation approach. Then, when the damaged road is detected through these networks' nodes, the location of it is published to SOXFire via SOX Node (sox-out in the flow).

The bottom half of the flow in Fig 13:

When multiple garbage trucks confirm an area of road damage, one truck is designated to upload a partial video of the area. A SOX-in node receives the location information, and the next node compares the location of the truck with it. If it is true, the driving recorder mounted on the truck sends the videos after anonymizing to the cloud visualization application. Before uploading, information related to people such as faces, or car matriculation plates is removed if the video contains them. Upload and display the videos in a suitable application for confirmation by city staff.

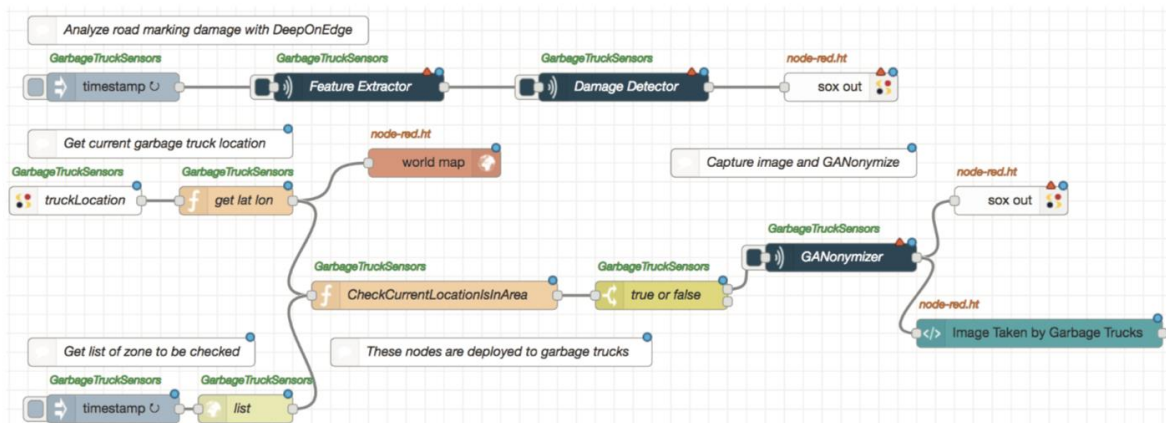


FIGURE 13: ROAD DAMAGE DETECTION APPLICATION DEVELOPED BY CITYFLOW

Thanks to Node-RED based programming in CityFlow, we released from engineering the API of web services, and also coding some snippets. This is very affordable for developers not to write similar snippets again and again. Simultaneously, thanks to distributed feature of CityFlow, we can manage the devices very easily. This is also very affordable that developers do not need to consider the situation or context of the devices installed to develop the distributed applications such as our road damage infrastructure scenario. So far development with CityFlow is achieved by us, we plan to involve information sector in municipality to understand how easy or difficult to use CityFlow in their daily city operation. In addition, we plan to deploy the flow in actual several garbage trucks in Fujisawa. These future works are addressed in coming deliverables of WP4.

3.5 Data analysis components

3.5.1 *System component - recommendation engine*

The Recommendation Engine developed by ICCS can provide recommendations to end users/citizens in several application settings such as transportation management, energy consumption, public safety, supply chain management, tourism services, air and sound pollution management, etc.

The Recommendation Engine consists of three independent components:

1. an IoT Node-Red flow component that wires together the different hardware devices, APIs and web services, connecting the distributed components and sensors into a common IoT application.
2. a Neo4j Graph Database;
3. the Recommender Application with a Front-End and Back-End which handles the interactions with the user;

IoT Node-Red Flows:

The first Node-Red flow, called the “Data Source Flow”, handles the input of the data from different sources. In order to effectively transmit data that are being exchanged between the sensors as well as the main system, the MQTT protocol is utilised.

Following the initial data processing, the information is being forwarded to the Management Flow, the goal of which is the control of the outcome of the recommendations (e.g. number/kind of suggested paths, type of recommendations, etc.). To achieve this, the system takes into consideration the provided users’ preferences and input. HTTP Requests have been implemented with the corresponding Node-Red nodes enabling user interaction with our system as well as with front-end applications (i.e. Mobile Applications, Web Applications, or any other type that can issue HTTP requests). This enables an efficient storing in the flow of parameters related to user preferences.

Neo4j Graph Database:

Using Graph Database Modelling as the reasoning technique of the tool, the system provides state-of-the-art smart city applications, based on open data captured by the city IoT infrastructure and user generated content (in terms e.g. of users' profile).

The modelling process and approach in graph databases can be regarded as equivalent to the approach of creating graph structures that reflect the queries we would like to answer. “Users” or “Ratings” for example comprise the nodes of the graph, “names” are the attributes of the nodes, verbs such as “likes” depict the relations that link the nodes, and whatever refers to such verbs is regarded as the attributes of the relations. An interesting way to model time in Neo4j is through the use of time trees. In this approach, nodes represent years, months, and days, etc. while every node contains an attribute “value”. By forming the time trees, we can link particular events or other measured data on these trees.

The implementation of the system is based on using Neo4j’s query language Cypher, a declarative, SQL-inspired language for describing patterns in graphs visually. Then, the model is populated by importing the application related data into the graph database linked to the suitable time nodes, adding additional integration layers when required e.g. an additional layer is required for wind level and speed.

A widely applied technique in collaborative filtering recommendation systems is identifying the closeness among different users with similarity metrics. To this direction, we model application users as vectors and then compute the distance between them. This way we locate users who are closer to the target-user, study their behaviour and make the corresponding recommendations.

Recommender Application:

The system provides a user-friendly way for users' registration in order to deliver recommendations. When a user is registered to the system, a new unique node is created with the apartment characteristics stored as properties. Afterwards, the user inserts preferences and historical data, and the corresponding nodes and relations are created in the graph database. The next step is the addition of evaluations regarding past and real-time recommendations. Every evaluation node is connected to the user who provided it, the corresponding node in the time tree. This way a specific rating is connected to all the related information of our model.

The system, taking into consideration these features, makes complex queries to a graph database to collect related information and produces personalized recommendations for the specific user.

Outcome - Recommendations:

The proposed recommendation service exploits similarity metrics among vector representations, users' preferences, previous recommendations applied and ratings to deliver in real-time recommendations. The service provides the most suitable recommendations after tracking the closest users, historical data, the number these schedules were applied and past evaluations. The user chooses the best one and afterwards optionally provides feedback in the form of a rating or suggestion for alteration if required.

3.5.1.1 Use in trials/pilots

Smart Mobility - Green Paths (Bristol):

The Smart Mobility trial will present air quality data (both long-term historic records and real-time monitoring) that will be collected from Bristol Open Data platform; latest measured levels from Bristol St Paul's air quality monitoring station data provided by the Environment Agency and data from the air quality IoT device provisioned by BIO. This multi-sourced data, including PM2.5/10, NOx, O3, CO2, etc., will be aggregated and input to the BigClouT data warehouse using the BigClouT application programming tool, based on Node-RED. Pollution will be measured on 6 locations of the city. Incidentally, a data visualisation dashboard for air pollution level will be shown in a public zone. For this implementation, KNOWAGE will be used to show the user air quality levels and trends.

The Recommendation Service will correlate air quality, weather data (temperature, wind and rain) and traffic data to suggest the "greenest" path for pedestrians; the idea behind this service is to support a web page that will display data about pollution as well suggested paths along which the quality of the air is good for outside activities. It should be noted that this application is transferable to the city of Fuijisawa, since a lot of environmental data are available for the area.

Industrial Zones (Grenoble):

A case identified by the Greater Grenoble City Area is the management of the different industrial estates managed by the Area, in terms of creating a community of actors in these zones and provide recommendations to the members of this community for events, restaurants, etc. The main idea is to create a mobile phone application that will bring together all different questions relating to users of the zone - transport options, dining options, information about the different

actors present on the zone, information about business events and various sport clubs, etc. The idea for this project was validated through a survey carried out with the users of the Inovallée zone who were overwhelmingly positive about the creation of an application.

The Recommendation Service will be used in this use case to provide personalised recommendations to the aforementioned end-users. Recommendations will be related to restaurants, food trucks, events, activities, etc. To generate recommendations, user profiles (preferences) and data about the restaurants, planned events, etc. of the area are needed. This data will become available through sensiNact and BigClouT data lake. Sources of data will be public websites, historical mobility data and generated events and mobile application usage, as well as real time data.

Based on the results of the recommendation, the service will also be able to provide suggestions about routes that the end-user could follow, building on top of the results of the green paths scenario. A trigger of alarms about potential events that happen may also be implemented.

Road infrastructure monitoring (Fujisawa):

The baseline of this scenario is already presented in section 3.2.5.1. Working on top of the results presented in that section (road damage detection and quantification), the Recommendations Service could support two functionalities: a. providing suggested (optimal) paths for the city agencies responsible for repairing road damage and b. making predictions about future progression of damage in certain areas.

3.5.1.2 Experiences

- The three components of the Recommendation Service (IoT Node-Red Flows component, Neo4j Graph Database, Recommender Application) were built as independent components with each other. This is an important feature for possible extensions and enhance scalability and programmability. Indicatively, in the Bristol and Grenoble scenarios presented, the application per se (and its interface) is provided by the pilot partners instead of the Recommendation Service, thus replacing part of the Recommender Application.
- It was confirmed that one of the most remarkable features of a graph database (its effectiveness into handling big volumes of information with real-time response to queries) is sufficient to support the use cases and their corresponding implementation.
- It has been realised that in all of these use-cases, the Recommendation Service can work alongside KNOWAGE; recommendations and data visualisation can be combined to offer higher level information to users.
- For the use-case of Bristol, due to the small amount of locations providing environmental data, a solution has been identified: extrapolating pollution data from other types of data (e.g. correlation of traffic data to high pollution values).
- It is of outmost importance to mention that, in order to train and populate the model implemented for the Bristol green paths use case, environmental data gathered by Tsukuba from the Fujisawa use-case (using StreamingCube) are used. Successful results from this attempt would be a proof of concept of the idea that data-sharing across cities indeed enhances the applications implemented for Smart Cities and supports meaningful cross-border collaboration.
- Similarly, the Grenoble use-case can be extended to other cities.

3.5.2 System component - KNOWAGE

KNOWAGE [1] is a suite that provides analytical and visualization functionalities. In the context of BigClouT project it is part of the City Data Processing module and, in particular, of its submodules Big Data Analysis and Visualization. Therefore, it is in charge to provide data analysis and data visualisation functionalities.

In order to perform analysis, KNOWAGE allows to connect to different kind of data sources, such as SQL and NoSQL databases. Moreover, it allows to retrieve data from other platform and systems, such as CKAN, or through the capability of interacting with REST APIs or of importing static files, such as CSV files. Data coming from heterogeneous sources can be combined in a common model by joining them on a common attribute.

Thanks to its built-in visualization features, it is possible to create dashboards to be used as a decision-making support for city officers. KNOWAGE provides the following widgets to be configured and used in a dashboard: Text, Image, Table, Html, Map (currently under development) and Chart. Deeping on the Chart widget, it allows to create different charts, for instance: Bar, Line, Heatmap, Pie, Radar, etc.

KNOWAGE's functionalities can be accessed through its GUI and, also, through the RESTful APIs [2] it exposes.

3.5.2.1 Use in trials/pilots

Road Infrastructure Management (FUJ):

The baseline of this scenario is already presented in section 3.2.5.1. KNOWAGE exploits its analytical and visualization capabilities providing a dashboard (Figure 14) that reports Fujisawa's road damage daily status. The input data are provided as CSV files containing the punctual measure of damage in the city gathered through DeepOnEdge deployed into the Garbage trucks. The analytical process, built for this use case, focuses on the aggregation of punctual measures by their geographical localization into three high-level features (Fujisawa's Districts, Fujisawa's Road and a Grid Mesh), the aggregated features are classified by their level of damage (High, Medium and Low). The high damage sorted output of the analytical process is provided to Fujisawa's Municipality and to Recommendation Service in order to, respectively, schedule the proper maintenance services and to suggest the best path to follow for the maintenance services.

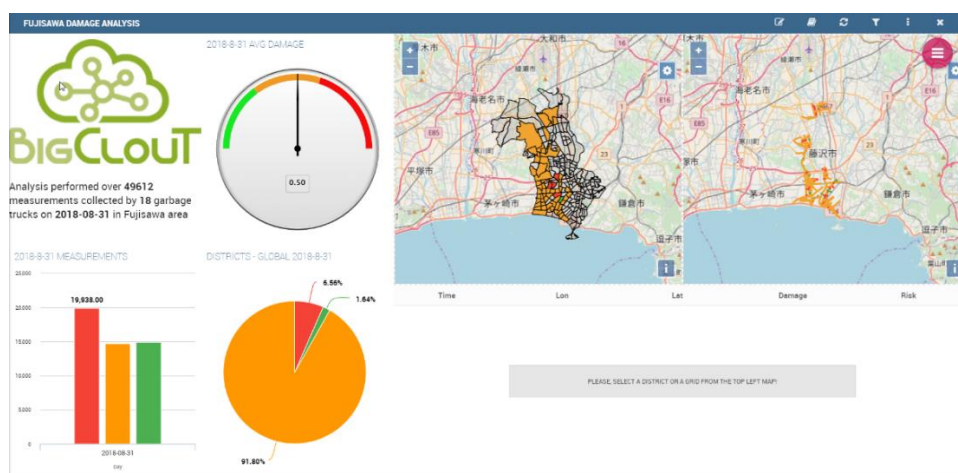


FIGURE 14: KNOWAGE - ROAD INFRASTRUCTURE MANAGEMENT DASHBOARD

Smart Mobility (BRI):

The baseline of this scenario is already presented in section 3.5.1.1. In this context, KNOWAGE is used to produce a dashboard that depicts air quality levels and trends stored into BigClouT Data Lake as open data leveraging also on outputs provided by the Recommendation Service (e.g.: showing statistics about the best green path suggested to citizens moving around Bristol).

Industrial zones (GRE):

The baseline of this scenario is already presented in section 3.5.1.1. KNOWAGE is used to visualize statistics about the usage of the mobile application. Moreover, it is used to visualize in a map the recommendation outputs from the Recommendation Service (e.g. restaurants or events) providing statistics about the most recommended place or event in the city.

3.5.2.2 Experiences

- Thanks to the several connectors it provides, KNOWAGE is able to interact with most of the others BigClouT's components. In particular, the integration with Recommendation Service is performed taking advantage of the RESTful APIs it provides. This integration allows to instantiate an environment able to perform two fundamental tasks related to decision support: generation of recommendation and data visualisation.
- Another important connection point is represented by the BigClouT Data Lake, in particular, KNOWAGE provides the specific CKAN connector. Although it is possible to retrieve data from the aforementioned asset through this specific connector, it is also possible to leverage KNOWAGE capability of interacting with generic RESTful APIs, using the REST connector; this represent an alternative way to interact with CKAN, providing the possibility to directly execute queries on CKAN.
- At time of writing, the integration of KNOWAGE with Edge Storage (section 3.5.5.3) is under investigation; the aim of this is to provide capabilities of reading and writing data from edge nodes thanks to CDMI interfaces.
- Finally, in order to provide different levels of details about road damages in Fujisawa's use case, the data analysis has been improved.

3.5.3 System component - adaptability framework

3.5.3.1 Overview

In the BigClouT architecture, a dependability and self-awareness framework are provided that maintains an up-to-date system context and adapt the platform configuration and behaviour in response to changes of the context. In Deliverable D2.3, the Dependability and Self-Awareness architecture, the generic self-awareness mechanism, and implementation are described and illustrated with a use case in the service composition feature of the platform. This use case addresses conflicts detection and resolution among applications developed in the sensiNact Studio and deployed in the sensiNact platform. To implement and demonstrate this use case, an ECA Adapter is developed as a plugin to the sensiNact Studio that is able to automatically detect and resolve conflicts among applications developed by one user. Furthermore, the core functions of the ECA Adapter can also be integrated to the platform to resolve conflicts among applications that are deployed in the platform that are developed by different users.

In the following two sections, the details of the ECA Adapter and its integration with the sensiNact Studio and platform will be described, along with our experience in the integration demo.

3.5.3.2 Use in trials/pilots/demos

Conflicts occur when two or more applications are able to perform different and usually conflicting operations on the same actuator or resource at the same time. Consequently, in such

situation the actuator or the resource can only satisfy one application's requirement at one time, resulting in unsatisfied requirements for other applications in conflict. The use case described earlier is to detect and resolve such conflicts for service composition. It is implemented as a plug-in, called ECA Adapter in sensiNact Studio and platform.

The ECA Adapter follows the same architecture described in Deliverable D2.3 on the self-adaptation for service composition, which is illustrated in Figure 15. The ECA Adapter is composed of three main components, namely the Translator, the Models@run.time Extension, and the Adaptation Engine.

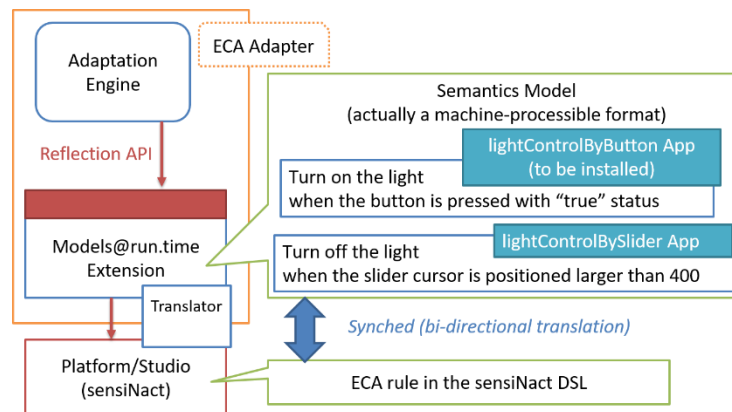


FIGURE 15 SELF-ADAPTATION FOR SERVICE COMPOSITION IN SENSINACT

The Translator provides a bi-directional translation between the sensiNact DSL (Domain Specification Language) and the semantic design model in the Models@run.time Extension. The sensiNact DSL specifies the Event-Condition-Action (ECA) rule for an application. For instance, Figure 16 illustrates an example of the ECA rules of the application lightControlByButton in the sensiNact Studio. Text in green explains the corresponding component of the ECA rule in the DSL.

```

lightControlByButton.sna
resource ON=[localgateway/light/switch/turn_on]
resource OFF=[localgateway/light/switch/turn_off]
resource button=[localgateway/button/switch/state]

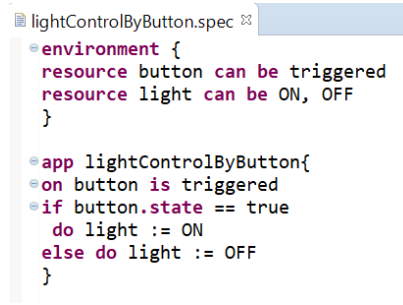
on button.subscribe() //Event: button is pressed

if button.get() == true //Condition: when the button is now with status being true
do ON.act() //Action: turn on the light
else do OFF.act() //(Condition-Action) otherwise turn off the light
end if

```

FIGURE 16 EXAMPLE OF AN APPLICATION'S ECA RULES IN SENSINACT STUDIO

When deploying such an application, the Translator translates the sensiNact ECA rule into a semantic design model that contains more semantic information of the implemented code in the platform, and with less implementation-level details, which is then stored in the Models@run.time extension component. For instance, Figure 17 illustrates the semantic design model for the previous lightControlByButton application rule shown in Figure 16.



```

lightControlByButton.spec
environment {
  resource button can be triggered
  resource light can be ON, OFF
}

app lightControlByButton{
  on button is triggered
  if button.state == true
    do light := ON
  else do light := OFF
}

```

FIGURE 17 EXAMPLE OF THE TRANSLATED SEMANTIC DESIGN MODEL

Based on the translated semantic models, a sub-module ECA verifier will check for conflicts between the to-be-installed application and those applications that are already deployed. If any conflict is identified, then the Adaptation Engine will be activated to resolve the conflict. Reasoning about the solution to remove a conflict is currently implemented by assigning priority to the conflicted applications and then injecting extra condition(s) to the lower priority application to make it more constrained to perform the action in conflict. The extra condition is the negate of the conflicted condition in the higher priority application. The Adaptation Engine will then generate a suggested update of the semantic model for the lower priority application; this suggested semantic model will then be translated back to the sensiNact DSL rule by the Translator. If no solution is found, a suggestion for stopping the deployment will be given with explanations to the application developer.

When the higher priority application is un-deployed, namely removed from the platform, then the previously in-conflict lower priority application will be suggested to roll back to its original rule, i.e. the added trigger, condition, and resources when resolving the conflict will be removed from the lower priority application.

3.5.3.3 Experience

The current demo did not implement a fully automatic (black-box) self-adaptation approach as we consider it important to involve the application developer in the update process. Thus, the developer will be prompted with a window dialog in the sensiNact Studio which shows the applications identified in conflict and the suggested rule, as shown in Figure 18. In the figure, the lightControlByButton application is already deployed, while lightControlBySlider is about to be deployed but a conflict is identified with the lightControlByButton application. In this way, the developer is made aware of the conflict and can choose to resolve it by clicking the "Auto-resolve" button in the dialog, then the content in the 'lightControlBySlider.sna' editor will be updated automatically. If the to-be-updated application is already deployed, to resolve the conflict, the application has to be re-deployed to the running platform after update.

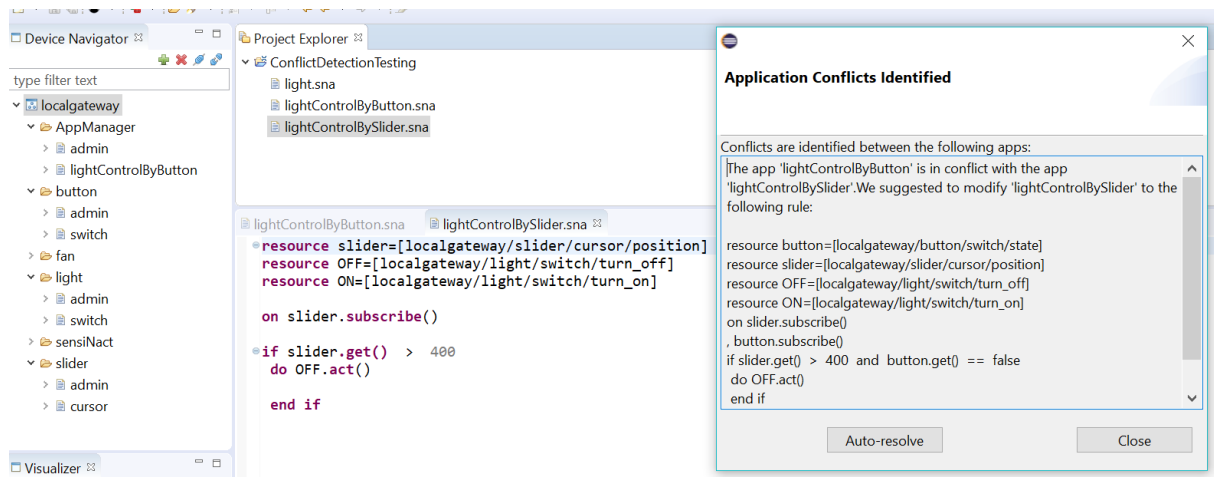


Figure 18 Application conflicts identified dialog for developers in sensiNact studio

The above screenshot is captured in the sensiNact studio when connecting to the platform, namely the 'localgateway' as shown in Figure 18. Those resources such as 'button', 'slider', and 'light', etc. are simulated devices in the 'localgateway' platform. This demo is an integration effort made between the sensiNact studio and the ECA Adapter which detects and resolves conflicts among applications developed by one user. As a plug-in, the ECA Adapter implements an extension where the Eclipse extension point is defined in the studio. When an IoT application is going to be deployed or un-deployed, the plugin will then be triggered.

At the time of writing, further integration between the ECA Adapter and the sensiNact platform is in progress to detect and resolve conflicts among applications developed by different users. One concern in this integration is the determination of which application should be considered as of lower priority since both developers can naturally consider their own application of higher priority.

3.5.4 System component - oneM2M Service Layer API

3.5.4.1 Overview

The oneM2M standard employs a simple horizontal, platform architecture that fits within a three layer model comprising applications, services and networks. In the first of these layers, Application Entities (AEs) reside within individual device and sensor applications. They provide a standardized interface to manage and interact with applications. Common Services Entities (CSEs) play a similar role in the services layer which resides between the applications layer and the in the network layer. The network layer ensures that devices and sensors and applications are able to function in a network-agnostic manner

An application connects to one of CSEs, then it can communicate to all of the CSEs and other applications connected to them via the routing capability of CSEs. Applications that collect sensor data then send to one of the CSEs allowing other applications to analyse the collected data. Analyses application can be adaptably managed by changing the location of the application and changing the hosting CSE. For example, for a CPU heavy analysis, it would be better to put the application on to the host which has enough capacity (location of AE 1 in Figure 19 Application locations in oneM2M System), and for the cases with big data transfer, it would be better to put the application near the data location, in particular, if the application and hosting CSE are located

on the same box, it can reduce communication overhead communicating directly through direct API calls (location of AE 3 in the Figure 19).

One of problems preventing this adaptation is that there are no standardised API for oneM2M applications. oneM2M allows multiple communication methods (CM), which are combination of protocols, includes HTTP, CoAP, MQTT and Websocket, and serializations, including XML, JSON, and CBOR. Application developers tend to write CM dependent code in oneM2M applications. Those applications loose portability and they can't communicate through CMs other than initially intended (Figure 20 upper part).

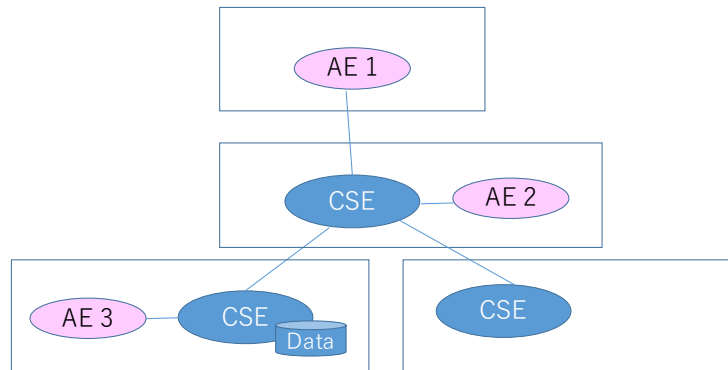


FIGURE 19 APPLICATION LOCATIONS IN ONEM2M SYSTEM

The Service Layer API for oneM2M and its implementation (Client Library) aims to solve the problem above (Figure 20 lower part and Figure 21).

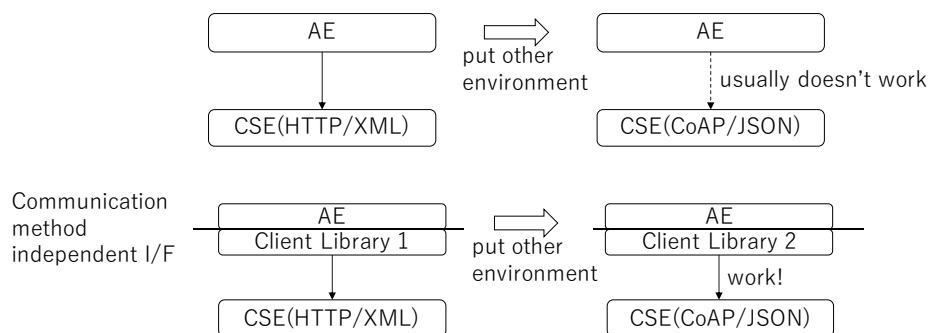


FIGURE 20 COMMUNICATION METHOD PORTABILITY BY API AND CLIENT LIBRARY

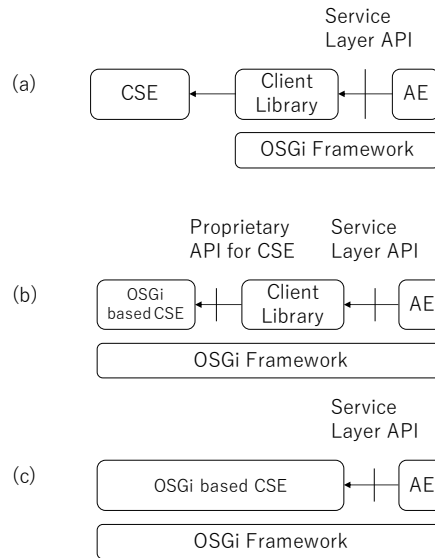


FIGURE 21 DEPLOYMENT TYPE OF AE AND CSE

In the OSGi Alliance, there is a standardisation work aiming to provide the API described above. RFP-187 “oneM2M Service Layer API” summarizes requirements for API and RFC-237 “Service Layer API for oneM2M” describes its solution. RFP-187 is agreed by the IoT Expert Group of OSGi Alliance and RFC-237 is mostly developed and stable but waiting feedback from reference implementations and compliance tests. Major features are CM agnostic, asynchronous supports, adaption of call-by-value, and support of both low level and high-level request. CM agnostic enables application as CM independent. Those application can run in another CM as long as client library for the CM is provided.

Asynchronous API enables applications to get immediate return of the call, even if the result is not ready, so they can do other operations while waiting for the result. Multi-thread programming is usually used with synchronous APIs to support this. Asynchronous programming can reduce the number of threads, it contributes to reduce resource and provide better execution performance. For realizing asynchronous API, OSGi Promise is used and it can be used synchronous invocation manner easily just putting a single line of code.

The Call-by-value concept is applied for the API, using OSGi Data Transfer Object (DTO), which is simple Java Object with only public fields of allowed simple classes and no additional methods and allows easy serialization. Call-by-value concepts reduce the potential problem in multithread environments. Data structures closed to root, which is request primitive and response primitive, are expressed as specific DTO, which has specific type and field name. Meanwhile deep structures are expressed as generic DTO, which has Map<String, Object> and accommodates key-value pairs.

The API supports both low level and high-level request. Low level request passes the data structure for request primitive and gets one for response primitive. Meanwhile high-level request is represented as CRUD operations of oneM2M resources. So, application developers can write descriptive code with high level API or write precise operations with low level API.

3.5.4.2 Use in trials/pilots

The specification is implemented and tested with the typical test cases, in our laboratory environment and it is proven that the API is implementable and works well. We have integrated the oneM2M Platform with SoXFire through Distributed Node-RED and demonstrated three types of applications. First application is 'I need Hurry App' application, it monitors location of a vehicle (For examples, commuting bus or garbage collection car) and reminds citizen in home to be aware that the car is approaching you. It can control both real device and emulated device using Node Red (Figure 22)



FIGURE 22 APPLICATION 1 I NEED HURRY APP

A second application is a data viewer application and it stores a series of historical data of some specific data (Figure 23). SoXFire can keep latest data and provide it to the client, but does not support historical data. Meanwhile oneM2M has a Data Management capability and it is suitable to store the series of historical data. The application is utilizing good aspect of both enablers.

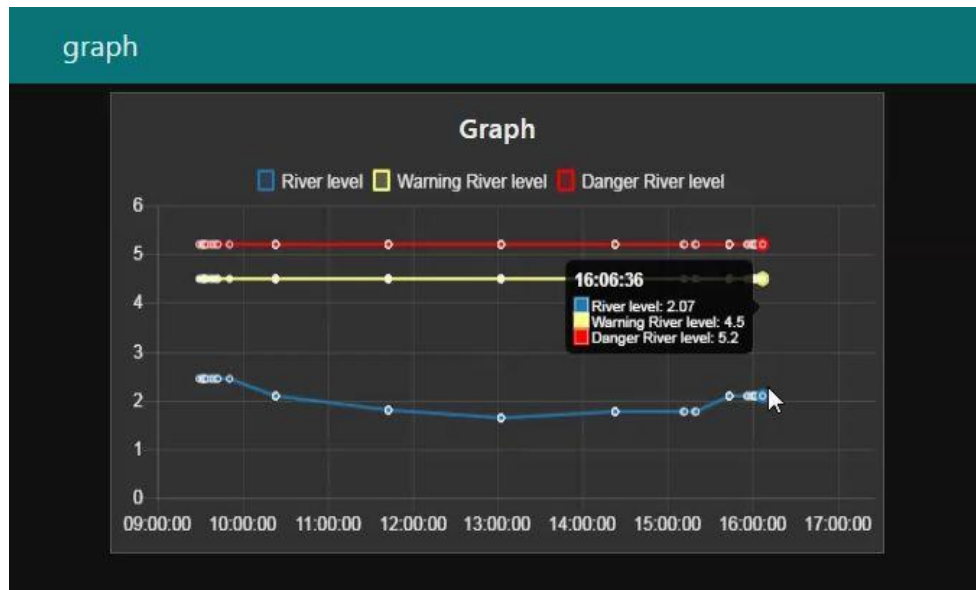


FIGURE 23 APPLICATION 2 INTEGRATION WITH SOXFIRE THROUGH D-NR

A third application is an opportunistic sensing application of Fujisawa city, it consolidates sensors on garbage collection cars and provides map view of sensor data (Figure 24). The application splits the map into segments and the latest value of the vehicle in the segment is regarded as the value of the segment.

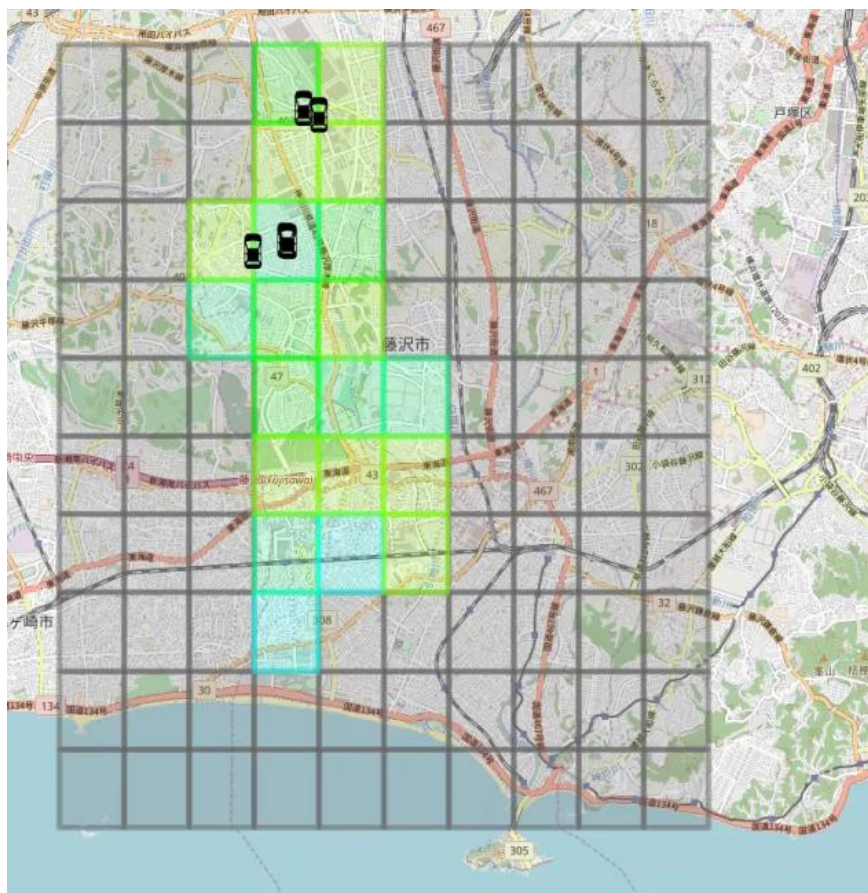


FIGURE 24 APPLICATION 3 OPPORTUNISTIC SENSING IN FUJISAWA CITY

3.5.4.3 Experiences

The oneM2M Platform provides a well-defined REST API, it is found that integration with D-NR is straight forward and very efficient using pre-installed HTTP Client node components. Data stored in oneM2M can be easily visualized. It is also efficient to implement smart city application using maps like application 1, using World-map-component, which are not pre-installed component, but it is registered common Node-red server and very easily installed.

It has also been found that oneM2M's data management capability is beneficial to storing and managing data. In the application 3, data in segments are stored to dedicated containers, which are easily managed to be displayed, handled and calculated. In the application 2 the data management capability is well complemented with SoXFire to provide historical series of data.

3.5.5 System component - CDMI Edge Storage

3.5.5.1 Overview

In IoT context, the edge of the network is populated mainly by smart devices and, more in general, nodes with low capabilities. Since the number of connected devices is increasing, the idea to share part of their processing and/or storage capabilities in an integrated System helps to manage a huge amount of data seems very promising. Edge computing tries to implement this idea: in BigClouT, besides computing, also storage concepts are declined on edge paradigm in order to share storage capabilities of edge devices in an integrated system. The Edge Storage System is composed by an arbitrary number of Edge Nodes, which, as requirement, need to host the CDMI Service for the interfaces and a data storage service. In general, nothing prevents to use smart devices as Edge Nodes, at the moment of writing this document, no version of the Edge Storage Service for smart devices has been developed yet.

Edge Storage paradigm enables to share limited storage capabilities obtaining a distributed storage. Cloud technology, indeed, provides potentially unlimited storage capabilities offered as a Service by hardware located on big data centres. The advantage of the former is latency: A Node, even if it is not a smart device, can be deployed everywhere and in particular close to the place in which data are produced and consumed. This is useful to store limited set of data to be immediately processed on the same place. The combination of Edge paradigm and Cloud technology combines high capabilities and low latency: these are the features of Fog Paradigm.

The Fog Storage is used to store historical data for future references on the Cloud and limited set of data on the Edge Nodes in order to be immediately. The Fog Storage is the System shown in Figure 25.

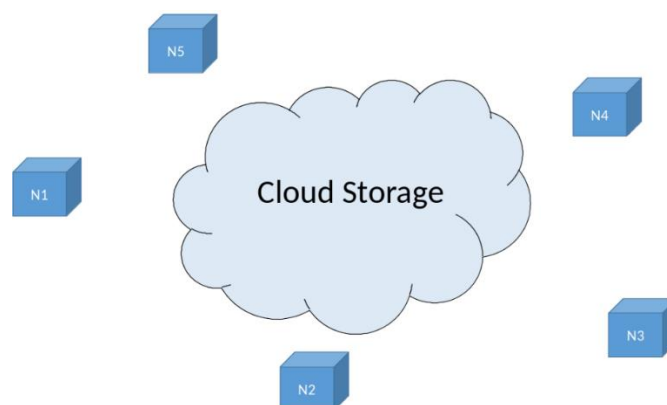


FIGURE 25 FOG STORAGE SYSTEM

The Cloud Storage has been conceived and implemented in ClouT project: the only specific improvement added during BigClouT concerns the support of nested containers. Besides this, all the new functionalities included concern the Fog paradigm and will be described later.

Figure 26 shows the details of the CDMI Cloud Storage: the Endpoint provides CDMI interfaces and the storage capabilities are implemented by integrating Hypertable and OpenStack Swift.

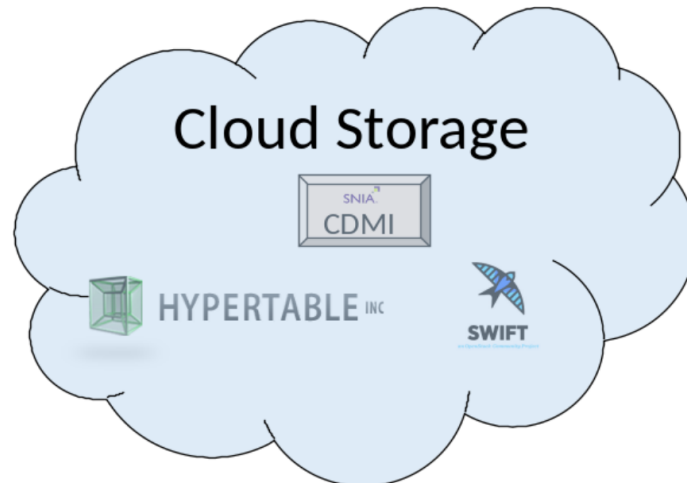


FIGURE 26 DETAILS OF THE CDMI CLOUD STORAGE

Edge Nodes provide the same interfaces compliant with CDMI Standard: for an external client is impossible to distinguish if the pinged endpoint identifies the Cloud Node or an Edge Node. Therefore, also Edge Nodes have an overall architectural schema including a service providing interfaces and an internal storage. In the current implementation, the internal storage used by the nodes is a relational database, in particular Postgres (Figure 27).

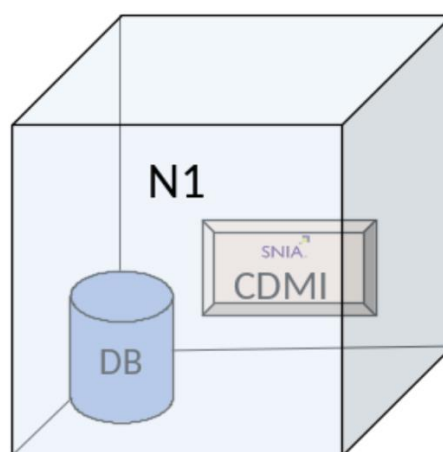


FIGURE 27 EDGE STORAGE DETAILS

As mentioned, both the Node types, Cloud and Edge, exposes the same REST methods compliant with CDMI 1.1.1 standard, in particular:

- *Container management*
- *Data Object Management*, included text data and byte stream (sensor data and object data)
- *Query Queues* (in the current implementation only local, i.e. Node based).

Besides the Cloud Data Management functionalities, the fog Paradigm includes the following features, implemented in BigClouT:

- Redirect mechanism: if a CDMI method is invoked on the Node on which referred data are located, the Node immediately sends the response acting as an autonomous element. Otherwise, i.e. if data are not on the same Node, a redirect mechanism is applied, in particular:
 - if the Node is an Edge Node, the request is redirect to the Cloud (Master) Node
 - if the Node is the Cloud Node, the request is redirect to the Node on which those data are stored (if exists): so, the Master Node keeps an index table including the information on where data are stored
- Data redistribution mechanism: if an Edge Node reaches the limit of its storage capabilities, part of its data is transparently sent to the Cloud
 - the data to be sent are, in general, taken from the container whose objects have been inactive (i.e. no GET, no PUT etc.) for more time.

The two described mechanisms provide both consistency to the whole Fog System and data proximity to sources and/or consumers. In particular, if the data of a certain container are continuously used (by multiple GET/PUT/POST operations), they are kept on the Local Node (until the storage space finishes), in order to have low latency. If the data are not locally needed any more and the local space finishes, a relocation to the Cloud does not impact on the overall latency. Based on these assumptions, the Fog Storage should be able to assure big capabilities and low latency.

3.5.5.2 Use in trials/pilots

The CDMI Cloud storage is part of ClouT architecture and plays an important role to store historical data to be maintained and acceded when necessary. The extension to Fog Paradigm provides also the capability to reduce latency. Data storage is one of the core functionalities that can be useful for all the planned pilots, at least concerning the main functionalities. Edge Nodes may be deployed in the cities, close to the data sources or data consumers, enabling to access data with low latency.

3.5.5.3 Experiences

The integration between the CDMI Storage and Sensinact has been completed during ClouT. Since the Cloud and the Fog Storage exposes the same interfaces, only some tests have been needed to assess the integration between Sensinact and the Fog Storage.

However, the in BigClouT, different paradigm has been adopted which affects the ability of Sensinact to manage the stored data: in particular the Cloud Storage was centralized and received data directly from Sensinact, which acted as dispatcher. In order to exploit the advantages provided by the Fog Storage, data producers and consumers, especially the ones located in the cities, should directly interact with the storage taking advantages from the standard interfaces. If an ad-hoc Node is deployed on a certain city and that Node is used to GET and POST only the specific data that are locally needed, the advantage in terms of overall latency will be significant.

At time of writing, the integration between the Fog Storage, especially concerning its Edge components and Knowage is in progress. This activity aims at obtaining the integration between big data analytics and storage: some tests have been successfully performed, others will be completed in the next weeks.

4 DISCUSSION & SUMMARY

One of the primary tools for programmability of smart city applications is the D-NR framework. This tool has been used in a number of the core city trials including the Fujisawa infrastructure sensing trial, the Bristol smart energy trial as well as a number of internal prototypes and demonstrations. Experiences with the tool can be divided into two main categories:

Usage for small scale data manipulation and distribution: in this class of applications, the primary use of the D-NR tool is to gather data, perform some basic manipulation, and post the data onto other services or data stores. In this usage model, the tool has been easy to use and provides a rapid development capability. Because of the large set of pre-built nodes, in particular the protocol and connector nodes, it has proven easy to quickly access and use data. An interesting use of D-NR has been as a ‘glue’ to connect together a chain of BigClouT components, e.g. taking data from the Orion Data broker, pushing it to the analysis component, StreamingCube, then taking the results and storing the results in the BigClouT data lake. In this class of examples, D-NR has shown the ability to both quickly develop small scale data flow applications, and also to quickly compose a set of BigClouT services to provide a programming chain.

Usage for larger scale city applications using CityFlow: Our initial experiences with using D-NR for larger scale applications highlighted the need to ‘break out’ of the D-NR environment to call and use external components. For some applications, this is acceptable and indeed wanted. However, some developers were interested in using D-NR for the full-scale development of the application. This led us to develop CityFlow and in particular to adding a set of AI nodes – the Tensor nodes described previously. Using this approach, we have been able to develop large scale distributed applications all within the D-NR tool.

As for the experience with the data collection and distribution platforms such as sensiNact and SOXFire, the performed tests and trials demonstrated one of the main features of BigClouT: easy and quick integration of data sources to the BigClouT data lake. SOXFire and sensiNact have demonstrated great capacity in terms of flexibility, scalability and extensibility for responding the requirements from specific trials.

With respect the BigClouT technology components, it is clear that the tools have met their goal of being modular and composable as has been shown by the number of trial applications that have been able to use the components to provide an end user service. Of particular note, the Knowage analysis and visualization tool has been used extensively throughout the project in a variety of trials. Equally, we have been able to show the use of the recommendation engine in a number of the trials.

While it is clear that the BigClouT architecture and tools meets the need for rapid and flexible programmability, there are some areas where further improvement could be made.

- Extensions to service composition. Both sensiNact Studio and D-NR have shown their ability to rapidly compose and deploy services. In addition, the demonstrators and trials have shown the ability to integrate and use the majority of the main BigClouT components. Of course, there is still room for improvement. In particular, some of the components, e.g.

the recommendation service still need work to tailor to trial specifics. This could be improved in a follow on phase if required.

- A key aspect of the platform is the ability to support dynamic adaptation. So far this has been used in small scale trials and clearly demonstrates the capabilities of the architecture and its integration with tools such as SenseNact. However, it still remains to fully integrate the adaptability framework and test fully with large scale application scenarios

Overall, even with the minor constraints noted above, the tools developed have demonstrated the programmability of the BigClouT architecture and the ability to rapidly develop a wide range of smart city applications, ranging from simple city dashboards, to complex streaming data analytics. While there is always room for improvement, the range of applications and the heterogeneity of the developers has clearly demonstrated BigClouT's ease of use and power.