B2S4B: A Platform for Smart City Business Processes Management and Adaptation

Emir UGLJANIN[1]*, Dragan STOJANOVIĆ[2], Ejub KAJAN[1], Zakaria MAAMAR[3]

[1] Department of Technical Sciences, State University of Novi Pazar, Novi Pazar, 36300, Serbia
eugljanin@np.ac.rs (**Corresponding author*), dr.ejubkajan@gmail.com

[2] Department of Computer Science, Faculty of Electronic Engineering, University of Nis, Nis, 18000, Serbia
dragan.stojanovic@elfak.ni.ac.rs

[3] College of Technological Innovation, Zayed University, Dubai, U.A.E
Zakaria.Maamar@zu.ac.ae

Abstract: This paper deals with the design and development of the B2S4B platform that bridges Smart City ecosystems (the business world) and city sensors: the IoT and citizens (the IoT and the social worlds). This platform allows the reconfiguration of Smart City business processes (BPs) based on the detected events and knowledge derived over time. In support of this, the concept of Smart City Observers (SCO) was introduced which acts as a man-in-a-middle between BPs, and the IoT and the social worlds by defining what should be monitored and how the respective observations could impact these worlds. In its turn, B2S4B analyses the detected events from all active SCOs and based on the predefined knowledge allows the synchronized reconfiguration of a BP and of the city sensors that feed the SCOs with accurate information. This paper proposes an approach for manual and automatic adaptation of Smart City BPs based on events reported by SCOs. In order to demonstrate the usability and effectiveness of the proposed approach this paper illustrates how this platform could be used in a case study on Smart City traffic control.

Keywords: Smart City, Business process, Internet of Things, Social Media.

1. Introduction

(Kourtit et al., 2012) state that in some European and Asian countries urban population exceeds 70% of the total population. These trends force Smart City (SC) development towards providing better services and wellbeing for urban population.

In line with the proposed research agenda, (Ugljanin et al., 2020; Kajan et al., 2020), a smart city is looked at as a set of mutually connected ecosystems, e.g. smart transportation, environment, homes, health, etc. Each of them is governed by some business processes (BPs) whose inputs come from the city sensors, namely citizens and things. Whilst the citizens could express their thoughts and observations via social media and smart devices participating in smart city initiatives, the sensors (e.g. cameras, temperature and air pollution sensors, etc.) provide their data to smart city ecosystems.

However, integrating Internet of Things (IoT) into BPs was not yet completely fulfilled. Some obstacles that undermine this integration are reported in the literature, for example in (Janiesch et al., 2020) and (Maamar et al., 2020), to mention just two such cases. In the former, among others, lack of standards, limited computational capabilities, security issues, and incompatibilities in process flows in a BP, and communication and data flows in an IoT environment are identified. In the latter, Maamar et al. (2020) identify IoT-related obstacles, namely silo restriction, the lack of IoT smartness and the lack of an IoT-oriented software engineering discipline that would guide the analysis, design, and development of IoT applications, and BP-related restrictions such as physical surroundings, data input, and context insensitivity. On the other hand, the "social fever" that drives people to express their thoughts and feelings via social media (e.g. Facebook, Twitter, etc.) has not been used as much in a business community as it should be expected due to power of data brought about by social media. Although Sánchez et al. (2022) state that people may feel social pressure to express true thoughts when anonymity is not guaranteed on social networks, Ugljanin et al. (2020) argue that by motivating citizens to contribute to community good results could be achieved.

This paper proposes the concept of Smart City Observers (*SCOs*) as a glue that brings together business world (smart city ecosystems BPs), social world (citizens as human sensors) and IoT world (things installed in a city). Their purpose is to help decision makers in the smart city to use the full potential of Social and IoT worlds, without the need for technical knowledge and in a very intuitive way. We design the Business to Social for Business (B2S4B) architecture and develop a platform based on *SCOs* with the purpose of establishing a meaningful communication between the three worlds mentioned above.

The main contributions of this paper are:

- Specification is provided and *SCOs* are implemented as modular, self-standing and event-based orchestrated services that sense and detect events on IoT and social worlds and perform the appropriate actions. These events are then collected and analyzed by *B2S4B* that improves smart city initiatives by re-configuring its BPs.

- A user-friendly tool, namely *SCO Designer*, is created for generating *SCO definitions* and *SCO chains* that allow non-technical users to create *SCOs*.

- *IoT Executor* is created, a tool that allows reconfiguration of IoT devices initiated directly by accessing IoT Executor or indirectly by sending the request from *B2S4B* platform.

- The B2S4B platform, its SCOs in action and constituent modules and components are evaluated in a case study focused on smart city traffic and citizens` mobility.

The rest of this paper is structured as follows. Section 2 presents related work along with background on social media, Internet of Things and smart cities. Section 3 sets forth the B2S4B platform, while the concept of SCO, including SCO definitions and SCO chain is described in Section 4. Section 5 presents the reconfiguration process, while section 6 describes the implementation of the B2S4B platform. Finally, section 7 sets forth the conclusion of this paper.

2. Related Work

Kaplan & Haenlein (2010) define Social media as "*a group of Internet-based applications that build on the ideological and technological foundations of Web 2.0, and that allow the creation and exchange of User Generated Content.*" Any individual with a smart phone and a social media account could be considered a human sensor that produces real-time data (Bauman et al., 2017). On the other hand Internet of Things can be defined as "*a world-wide network of interconnected objects uniquely addressable, based on standard communication protocols.*" (European Commission for Information Society and Media, 2008). It is expected that SC utilizes its computational and communication resources, analyses, manages and integrates huge amounts of generated data to increase safety, efficiency, productivity and the quality of life for citizens (Gharaibeh et al., 2017). Data collected from IoT in a business process enables automated management and time-efficient decision making (d'Hondt et al., 2019).

There are several available platforms that support smart city initiatives and use IoT and Social related data separately or combined. For example, FIWARE is a cloud-based IoT platform that provides a set of APIs employed for developing smart applications. It is a project sponsored by the European Commission which is based on previous SENSEI project and IoT-A architecture that contains components named Generic Enablers. Those components are used as building blocks for smart applications (Araujo et al., 2019).

Another cloud-based platform capable to process heterogenous data coming from multiple sources is presented by Chamoso et al. (2020). It reuses software and other platforms in order to simplify smart city application development process. Although this approach is appealing, it requires technical knowledge to be able to integrate such components in the smart city applications.

CityPulse (Giatsoglou et al., 2016) is a platform which uses near real-time IoT data and social media. Although CityPulse enables cross-domain data integration, their approach of analysing social media streams from Twitter focusses on analysing content using machine learning. They consider all streams on a certain subject which can be misleading.

On City Data and Analytics Platform (CiDAP) platform (Gavrilović & Mishra, 2020), IoT data is acquired by special IoT agents and IoT brokers, whilst data from social world is saved in Hadoop Distributed File System (HDFS) and Spark applications for further analysis. IoT agents are then made available to smart city applications by an CityModel Server, but there is no evidence of IoT agents cooperation.

In (Faci et al., 2018) bidirectional communication with IoT devices is examined in terms of mutation of ESP8266 IoT devices Over The Air (OTA), and metrics for evaluating social actions in a form of citizens' engagement are presented in (Kajan et al., 2020).

This paper presents a circular process of BP adaptation by reconfiguring SCO that may or may not produce new data and change the way how BP, social and IoT world are affected. In (Ugljanin et al., 2017) the connection between business processes and social networks is proposed, where business process can initiate some social action on a social network and depending on users' responses decides what tasks should be performed further.

3. B2S4B Platform

Chamoso et al. (2020) state that Smart City platforms must overcome problems related to the large amount of collected data which is categorized into scalability and reusability.

The proposed B2S4B platform provides a solution to these problems. It is a highly flexible solution for easy and smart management of city assets, such as traffic lights and parking system. It supports business processes adaptation based on detected events, such as traffic jam, and allows city managers to monitor and make an impact on the Social and IoT world without the necessity of having technical knowledge. *B2S4B* is a suite of microservices that are loosely coupled, distributed, capable to stand alone, and run as separate services communicating via publish/subscribe or HTTP protocol (Neuman, 2015). Each microservice can be deployed within a separate container and scale according to the requirements. Several data-related issues were considered while building this platform. Firstly, it is important to find an efficient way to exchange unstructured (social world data) and semi-structured data (IoT data). Secondly, one should find a way to process both real-time data and historical data on demand and automatically. Finally, it is necessary to find the best storage solution for storing a massive amount of data generated by IoT devices and social networks.

The *B2S4B* platform architecture (Figure 1) consists of three main modules, namely *Connectors, Smart City Observers* (*SCOs)* and *B2S Manager*, along with components that connect them and provide their interactions, such as repositories and message brokers.

Connectors contain services that are acting as middleware between *SCO* and both the Social world (social media such as Facebook or other crowd-based apps) and the IoT world (sensors and actuators, such as traffic lights, weather service, air quality, etc.). They provide data collection, processing and storage capabilities and can feed *SCOs* with more refined and historical data on demand. *Connectors* possess bidirectional communication with Social and IoT worlds, that includes collecting the data from

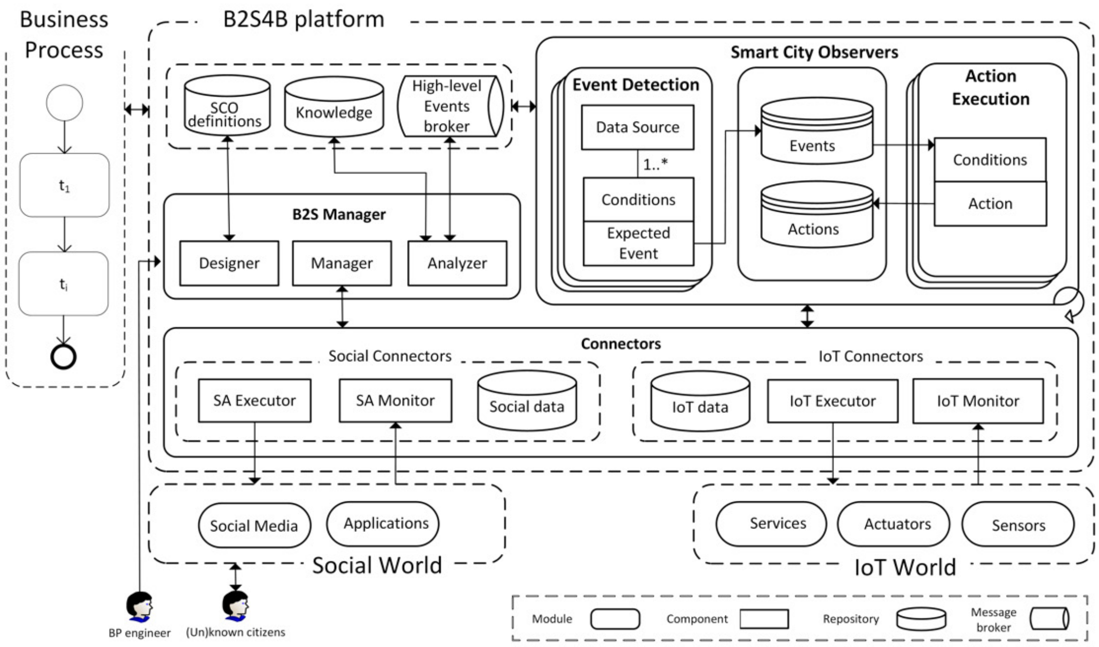

Figure 1. B2S4B platform architecture

both worlds (e.g. retrieving data from Facebook campaign, or reading temperature sensor value) as well as actuating on it (e.g. publish a new campaign on the Facebook, or change a traffic light). Collected data is provided to *SCOs* in two ways, namely in continuous streaming fashion (e.g. feeding data from sensors every second), and as a response to a request (e.g. retrieve number of check-ins at some place via Facebook). In the case of continuous monitoring, data is published regularly so that all *SCOs* that are interested in it could subscribe to it. When it comes to providing data on request, the result depends on the time when SCO's request is sent and metadata is provided. This means that the same request sent in different moments could produce different results, as in the case of sending different metadata at same time. Based on the request and nature of collected data, the *Connectors* may or may not correlate the data, and build more meaningful and enriched data (for instance, a social flow for data gathered from social media (Ugljanin et al., 2016), or a data cluster for IoT-related data), prior to applying desired processing methods.

The aim of this paper is not to build new metrics for evaluation of data from either Social world or IoT world, but just to use ones that were already developed in the previous work. Depending on its nature, communication between *Connectors* and *SCOs* could take place via any communication protocol like HTTP or publish/subscribe-based protocols, such as MQTT.

Social Connectors contain two components, namely *SA Executor* and *SA Monitor* and one *Social data* repository. The role of *SA Executor* is to initiate social action, whether on social media or a smart city Web application/portal (e.g. publishing a campaign on Facebook, driver alert via mobile app about changes in the traffic, etc.). *SA Monitor's* purpose is to retrieve citizens' responses from Social World. Gathered data could be processed in real time applying NLP or similar techniques such as detecting emotions or applying sentiment analysis. Retrieved social responses could be annotated with useful information (such as positive or negative reactions for instance) for use in postprocessing stage and are stored in the *Social data* repository (Kajan et al., 2020).

Analogously, *IoT Connectors* contain two components, namely *IoT Executor* and *IoT Monitor*, and one *IoT data* repository. *IoT Executor* allows controlling IoT devices (namely sensors and actuators) by either sending requests for actuation (e.g. open or close the parking gate) or by reconfiguring its software that is underlying those devices by adding new capabilities. *IoT Monitor* allows collection of data from IoT devices and unifies the format for messages that need to be consumed by appropriate *SCOs*. The received data can be processed and is stored in *IoT data* repository.

Smart City Observers include two components, namely *Event Detection* and *Action Execution*, and two repositories, namely *Events* and *Actions*. *Event Detection* analyses data gathered from input such as *Connectors* or another *SCO* and tries to identify events based on *SCO's* conditions defined for each input. Detected events are stored locally in the *Events* repository at *SCO's* side and published via message broker for advanced analysis by *Analyzer*. *Action execution* analyses detected events and takes proper actions as per defined conditions from *SCO definitions*. Executed action is in fact an event that will be reported to *Analyzer* and stored in *Actions* repository.

B2S Manager allows at first, BP engineer to manually create/update *SCO definitions* through *Designer* component and manage *SCOs* through *Manager* component. It also allows automatic reconfiguration of *SCO definitions* and automatic management of *SCO* through *Analyzer* component by applying rules defined in the *Knowledge* repository based on detected *High-level Events broker*.

4. Smart City Observer

This paper proposes a Smart City "covered" with a network of *Smart City Observers* (*SCOs*) that could be self-standing or a part of *SCO chains*. A *SCO* represents a self-contained, temporarily constrained, planned and autonomic Smart City activity that is driven, adapted and reconfigured based on detected events and defined conditions. *SCO* monitors different sources of data to collect data and evaluate it with the aim of detecting events (e.g. high traffic detection: by monitoring street traffic sensors). Detected events trigger specific actions based on defined conditions such as publishing new social action on social media, reconfiguration or actuation on IoT devices, and sending a message to another *SCO*. Questions that one should ask when trying to identify or design a *SCO* are "What should be monitored?", "Which events are expected?", and "How to respond based on the detected events?".

Each *SCO* has at least one set of *Event Detection* and *Action Execution* components that defines its life cycle as well as *Events* and *Actions* repositories to store detected events and executed actions. Figure 1 shows *SCO's* components and its properties as well as a data flow. *Events* and *Actions* repositories do not persist after *SCO* lifecycle is finished.

4.1 Event Detection

A *SCO* could track one or several sources in parallel with different conditions and expected events, thus a *SCO* could have multiple independent *Event Detection* instances. Each *Event Detection* instance is defined with *Data Source*, *Conditions* and *Expected Event* properties. *Data Source* represents a source that is being monitored, such as another *SCO* which is monitored directly, or IoT/Social World which is monitored indirectly. A *SCO* has two ways of monitoring data from sources, either by subscribing to a topic or by sending requests in defined time intervals. In case that requests are sent, they contain reference (e.g. ID of social campaign to be tracked on Facebook) or instructions for analysis to be applied (e.g. specific metrics for evaluating a social campaign). This way the same data source could provide different data depending on the request and it is not exclusively reserved to one *SCO*. *Conditions* include rules that are checked in the process of event detection. Knowing that the same event detection conditions used in different contexts could be employed for detection of different events (e.g. 100 cars per minute is low traffic for highways, and high traffic for a one-way street), for each instance of Event Detection component a different condition is defined. *Expected Event* is semantically meaningful and could be used in the process of reconfiguration or action execution, for instance "traffic jam" or "normal traffic".

Each *Event detection* should: (1) indicate whether it monitors another *SCO*, or data source from Social or IoT World, (2) define time interval during which certain events should be expected, (3) provide metadata in case that a request is to be sent (e.g. metrics to be applied to a monitored campaign), and (4) define which events are expected and what are the conditions for their detection.

Upon event detection, the event is stored locally in *Events* repository on *SCO* side, as well as broadcasted via message broker as "High-level event" to the *Analyzer*. Local events are accessible to origin *SCO* only and persist if origin *SCO* is still active. "High-level events" on the other hand do not expire and are stored permanently outside of *SCO*. They are represented in the form of a quadruple sequence with 4W (who, what, when and where)

```
<Source, Event, Temporal, Spatial>
```

structured and formatted in JSON-LD to allow better readability and machine processing. *Source* links to the source that detected the event (e.g. traffic camera). *Event* refers to detected event, from the list of predefined events along with additional metadata that could be valuable for that event and is related to the event. *Temporal* contains the date and time when an event is detected. *Spatial* refers to the geolocation where an event is detected.

4.2 Action Execution

The purpose of this component is to execute actions on IoT and Social World via connectors which could be the same/different from the one used in the *Event Detection* instances and/or interaction with other *SCOs*. A *SCO* could have multiple *Action Execution* instances and overall number of its instances does not need to comply with the number of *Event Detection* instances.

Action Execution component relies on the events previously detected in the process of *Event Detection* and evaluates rules (*Conditions* property) for executing a specific action (*Action* property). Different actions could be executed based on different conditions and could convey additional data, such as a text used for publishing on social media, or a script to be uploaded on an

IoT device. *Action Execution* component instances could have different priorities if executed at the same time.

Although the direct result of this module is the action execution, there is a side effect of this process, which is producing a new event (e.g. changing the traffic lights signal timing in a particular street is indeed an event that another *SCO* could monitor).

Action execution should define: (1) conditions that are applied onto detected events and that will be used as a trigger for an action, (2) a side-effect event that is the result of executing an action, (3) subject of the action (another *SCO*, IoT or Social World that will be used for action execution), (4) what will be conveyed to the subject in terms of metadata, (5) the frequency of action execution and whether consecutive actions could be executed, and (6) scheduled time for action execution.

4.3 SCO Definition

To allow non-technical users to create and run *SCO* the *SCO definitions* specification was created, which is reusable, extendable and shareable, and can be used for *SCO* instantiation within *B2S4B* platform. *SCO definition* does not contain any code, and it does not define which programming language will be used for its implementation.

The JSON-LD was employed for formalizing *SCO definitions*, and an example is presented in Listing 1. It represents minimal *SCO definition* with one *Event Detection* and one *Action Execution* instance. It monitors camera at "Avnoja" street via *IoT connector* in order to detect a "traffic-jam" event. If the "traffic-jam" event is detected for an hour, an action (reconfiguration of IoT device) should be executed that will result in a change of the traffic lights system. *SCO definition* identification number is stored in *_id,* which will be used during *SCO* initialization. The *eventDetection* part defines 'Event Detection' module, and *actionExecution* 'Action Execution' module.

Listing 1. An example of a simple SCO definition

```
{
 "_id": "a061e6983a464911b92b297371055254",
 "description": "Monitor traffic",
 "eventDetection": [
   {
    "name": "street",
    "sourceType": "iot-connector",
    "location": "avnoja",
    "date": {
      "from": "2022-05-07 00:00:00",
      "to": "2022-06-06 00:00:00"
    },
    "description": "Monitors traffic in Avnoja street",
    "expectedEvents": [
      {
       "conditions": {
         "all": [
           {
            "fact": "origin",
            "operator": "equal",
            "value": "street"
           },
           {
            "fact": "location",
            "operator": "equal",
            "value": "avnoja"
           },
           {
            "fact": "value",
            "operator": "greaterThan",
            "value": 0
           },
           {
            "fact": "value",
            "operator": "lessThanInclusive",
            "value": 20
           }
         ]
       },
       "event": {
         "type": "traffic-jam"
       }
      }
    ]
   }
 ],
 "actionExecution": [
   {
    "conditions": {
      "all": [
        {
         "fact": "events",
         "operator": "detected",
         "value": {
           "type": "traffic-jam",
           "freshness": "an hour ago"
         }
        }
      ]
    },
    "event": {
      "type": "action",
      "params": {
        "id": "1",
        "name": "traffic-light-emergency-mode",
        "description": "Change traffic lights",
        "connector": "iot-connector",
        "frequency": "once in an hour",
        "allowConsecutive": false,
        "date": {
          "from": "2022-05-07 00:00:00",
          "to": "2022-06-06 00:00:00"
        },
        "content": {
          "nodeid": "4540533",
          "codeid": "11"
        }
      }
    }
   }
 ]
}
```

The source to be monitored is defined with *name* and *location*, *date* defines timeframe during which it should be monitored, and *conditions* define which rules should be fulfilled so that the expected event is detected.

The same goes for defining rules for action executions, except that *event.params* define where the action should be executed, how often it should be executed and if it will be repeated in case it was already executed. *event.params.content* contains metadata that define reconfiguration code with ID 11 which should be executed on device with ID 4540533 once traffic jam is detected and will be executed only once via IoT connector.

4.4 SCO Chain

SCO and BPs do not stand in contrast to each other. The main difference is the nature of artifacts used.

On the one hand a BP acts upon business artifacts that are defined as "*a concrete, identifiable, self-describing chunk of information that can be used by a business person to run a business*" (Nigam & Caswell, 2003).

On the other hand a *SCO* deals with social artifacts that may be seen as "*a form of online activity (e.g. tweet, tag, and endorse) that occurs over an open, Web 2.0 application*" (Maamar et al., 2015), but also with IoT artifacts such as videos taken by smart camera, data emitted by wearables, and actions triggered by captured sensor's signals.

When a *SCO* is looked into from the perspective of a business process management, a *SCO* may be seen as a task or subprocess, while a chain of *SCOs* may be seen as a part of a BP or whole BP.

Figure 2 illustrates an example of SCO chain composed of five SCOs, where each SCO has a different number of inputs and outputs, interacting with Social and IoT worlds and other SCOs. The chain starts with SCO1 receiving the input from Social and IoT world, and ends up with SCO5 executing an action on Social and IoT worlds. In a SCO chain each SCO runs independently, so that any SCO instance could be replaced with one or more SCOs without the need to stop other SCOs. SCOs in a SCO chain can easily be reconfigured (changing their duties and actions) or scaled (powering it with the additional computer nodes if needed). This way a modular system with independent parties was obtained, which is easily scalable and extendable. It is possible to decompose any Business Process and identify SCOs that represent its parts just by asking the questions for SCO identification, mentioned in section 4 and implementing BP logic by the SCO chain.

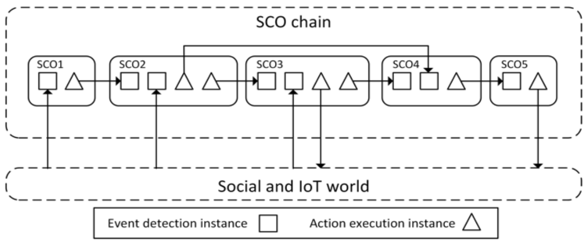

Figure 2. SCO chain

5. Reconfiguration Process

SCO's Event Detection deals with expected events only, which is great in many situations, but a problem arises when there are unplanned situations such as the need for fast reaction to solve parking problem during city mass event or proper reaction in cases of some natural disasters. Instead of spending a lot of time to design a new process, it is more efficient to reconfigure the *SCO* of interest in a *SCO chain* or to replace it with a new *SCO* designed to overcome an unwanted situation.

This paper considers three types of reconfigurations that could take place. The first one includes adjustment of rules for event detection only, where the action remains the same (e.g. changing threshold for traffic jam, from 100 cars per hour to 200 cars per hour). It does not affect environment and could safely be done in any process. The second reconfiguration includes adjustment of action that will be executed upon event detection (e.g. reconfiguration of IoT device, new action on social media, etc.), and has an impact on the environment and should be done carefully. Finally, the third reconfiguration is applied to the whole *SCO* that is in the chain. It could involve replacing a *SCO* with one or more other *SCOs*.

Reconfiguration process could be started manually by BP engineer through *Manager* or automatically by *Analyzer*. In either case, changes to *SCO definition* are required, the only difference is who will trigger *SCO definition* change. After the *SCO definition* is changed, the *SCO* initiated with it will be notified to reload and then deal with the new situation. In case of replacement, the old *SCO* will be stopped and replacement *SCOs* will be initiated.

Automatic reconfiguration is related to *Analyzer*, *High-level Events broker* and *Knowledge* repositories. *Knowledge* repository contains rules that will be consulted prior to starting automatic reconfiguration by *Analyzer*, and *High-level Events broker* repository contains all the events ever detected by any *SCO*. So *Analyzer* has an overall, global look at the situation in the city, while *SCOs* are local and granulated.

In the example below, the knowledge rule does not take into account who is detecting events as long as the events are detected, thus it gains helicopter view and a better insight in what is happening in the city at a higher level.

Knowledge rule example – If "crowded event" is detected in "city center", "Avnoja street", "Prvomajska street" during "past hour" , the traffic system should be adjusted and "mass event" is detected.

Although knowledge rules do not consider *SCO* that detected events of interest, data origin is important in terms of trust as different sources could have different weights (e.g. an event detected based on the data collected from IoT is more trustworthy than one which is based on data from Social media). That way, a "traffic jam" event in the rule above, is a universal event and is not related to a specific *SCO* or a specific type of device.

Reconfiguration is a part of BP reengineering that handles BP lifecycle (Dumas et al., 2013) in a continuous loop between „as-is" and „to-be" stages allowing decision makers to provide best services to citizens and keep a BP accurate in achieving business goals.

6. Implementation

This section presents a case study related to the implementation of B2S4B platform, the technical challenges encountered during this process, and B2S4B platform in action.

6.1 Case Study

Smart City needs to capitalize on data gathered from social networks and IoT and allow automatic but controlled adaptation and reconfiguration of business processes. The purpose of business process adaptation is to provide the citizens with the best services.

For illustration purposes, the city of Novi Pazar was chosen, which organized the event "Dan dijaspore" for the first time. Due to historical heritage and geographic location limitations, the city lacks big squares, so organizing mass gatherings would require closing streets and changing traffic system. The Smart City does not have an idea how many people to expect and what is the optimal solution for the event. It wants to react according to citizens` interest and to avoid closing too many streets if unnecessary or avoid closing them too early.

The idea is to promote the event and check citizen interest by monitoring how many people plan to attend the event by reporting on Facebook and how they react to the Facebook campaign that announces the event. If high interest in the event in detected the city decides to close Avnoja street for the concert and starts monitoring crowd gathering 10 hours before the concert starts to determine street closing time. If the crowd starts gathering it will close Avnoja street for new traffic and citizens will be informed about it via social networks. Closing the Avnoja street is a trigger to starting monitoring 3 nearby parking lots to check if the citizens coming to the concert have enough place to park. If all the parking spots are busy, the city will close "Stevana Nemanje" street for traffic transforming it into temporary parking and alerting citizens on social networks.

6.2 SCO in Action

For this purpose, 3 SCOs were created and deployed. *SCO 1* monitors Facebook event interest and citizen's responses on the Facebook Campaign to detect interest in attending the concert. If high interest in the event is expected *SCO 2* will close Avnoja street and alert citizens via Social media, but only when the crowd starts to gather. Closing Avnoja street is a signal that the event is starting and *SCO 3* starts monitoring parking availability in surrounding parking lots. If there is no parking available, it will close "Stevana Nemanje" street and notify citizens about the new available parking spots. Figure 3 shows a *SCO chain* that is automatically generated with the proposed *Designer* component based on the created *SCO definitions*.

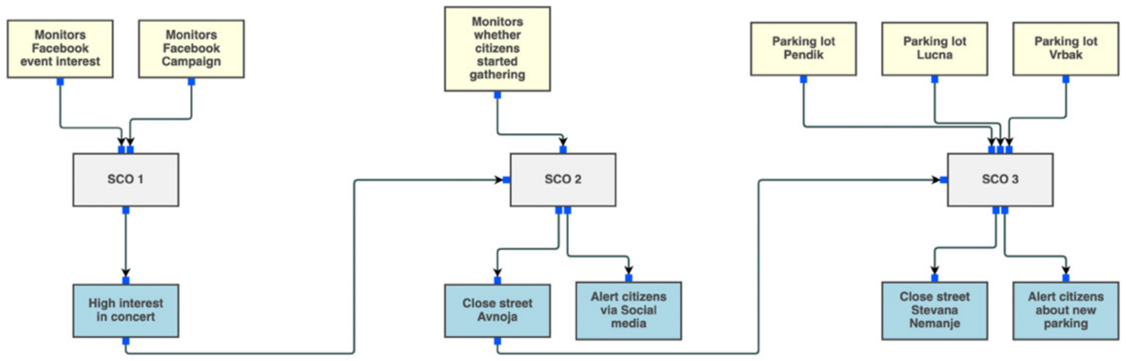
Monitors Facebook event interest
Monitors Facebook Campaign
SCO 1
High interest in concert
Monitors whether citizens started gathering
SCO 2
Close street Avnoja
Alert citizens via Social media
Parking lot Pendik
Parking lot Lucna
Parking lot Vrbak
SCO 3
Close street Stevana Nemanje
Alert citizens about new parking

Figure 3. A SCO chain created using the Designer component

6.3 Technical Challenges

When it comes to monitoring IoT and Social world the "What if" simulation approach is employed (Banks, 1999) in order to simulate different data sources allowing control of their values through the dashboard that helps to test different real-life scenarios.

Data collected from these sources is further provided to B2S4B platform through the respective Connectors. Real "Wemos D1 mini" ESP8266-based chips are also employed to show how reconfiguration could be made using the proposed open-source *IoT Executor* (GITHUB, n.d.-a) by mimicking traffic light system.

The proposed implementation allows the user to: (1) create *SCO definitions* and *Knowledge*, (2) create a *SCO chain* and display its graphical model, (3) manage *SCOs*, (4) simulate Sensor's readings, and (5) make an impact on Social and IoT worlds indirectly.

SCO definitions are stored in MongoDB database to be used in the process of *SCO* instantiation. To create *SCO definitions* one uses an in-house user-friendly dashboard based on JavaScript and PHP hosted on connect.rs (Anon, n.d.-a). *SCO chain* is displayed with GoJS graphic library (GOJS, n.d.).

Docker containers (DOCKER, n.d.) are used for running *SCOs* and container management tool Portainer for managing containers and *SCO* images (Portainer, n.d.). To instantiate *SCO*, SCOID (identification of respective *SCO definition*) should be provided as environment variable based on which the appropriate *SCO definition* is fetched and *SCO* starts executing tasks that are assigned to it. NodeJS is employed for running *SCOs* inside containers. *SCO* Conditions are evaluated using JSON Rules Engine (GITHUB, n.d.-b).

Sensor simulation dashboard allows users to control parking space availability in three parking lots, crowd gathering in Avnoja street, and simulate citizens' attending Facebook event and interest in the concert over social networks. Controls on the dashboard enable users to adjust parameters and check different conditions and how the system responds. Open-source message broker Mosquitto (Mosquitto, n.d.) was installed and configured on a Linux Ubuntu server to allow communication between *B2S4B* modules. It supports MQTT (MQTT, n.d.) as a fast and lightweight message exchange protocol based on the publish/subscribe pattern to allow communication between the proposed simulation dashboard (Social and IoT worlds) and *SCOs*. Connector simulation dashboard is publicly accessible via (Anon, n.d.-b). The IoT world reconfiguration is demonstrated by creating a model of traffic lights system. It consists of a "Wemos D1 mini" device wired with appropriate led lights, connected to an external battery and configured to use IoT executor framework. By changing the values in the Simulation dashboard, different data is provided to running *SCOs* which make changes to traffic lights system.

High-level events are reported to *Analyzer* using Kafka broker, and *Analyzer* uses Siddhi complex event processing engine to process all the events detected and executed by all available *SCOs*.

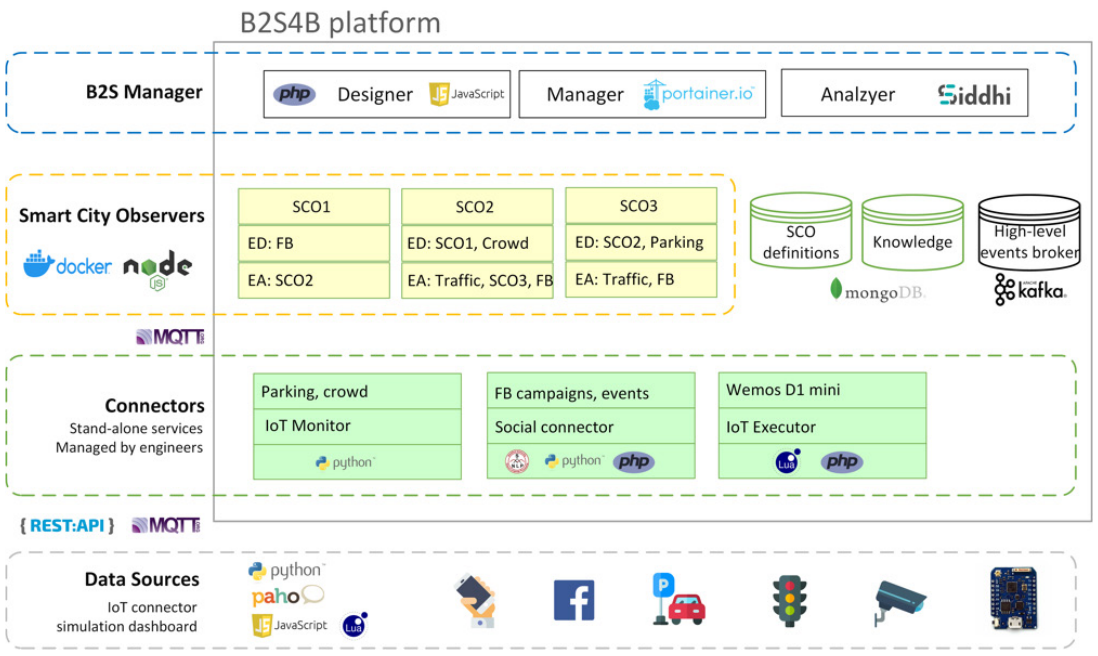

Figure 4. Technology stack for the implementation of B2S4B

Implemented components, underlying technologies and communication protocols are shown in Figure 4.

6.4 Scalability and Reusability

Platform scalability was analysed from four perspectives: (1) decision making actors; (2) scaled components; (3) communication between *B2S4B* platform components and (4) insights on expected volume of data exchange between components.

The platform supports local and city-wide decision making. Local decision making is related to a *SCO*, that in fact monitors a small area in a Smart City (e.g. a street, an intersection, or a tunnel). As such, a *SCO* will monitor only a few data sources, indirectly via Connectors, that way it shall collect data only from relevant data sources. This way the need for high CPU power is reduced for machines that run *SCOs* and one can save network bandwidth when it comes to simple events detection and reacting to them.

In the Smart City case study, one proposes the deployment of IoT Connectors in the Fog layer close to the IoT devices they communicate with, where each Connector is assigned only to a specific area of the city. Each Connector is in fact a data sink and uses a different broker to communicate with related *SCOs* and data sources. Thus, by adding more Connectors horizontal scaling is allowed. This way network load is distributed and specific protocol limitations in terms of allowed number of clients, data volume, or number of messages exchanged are avoided. For instance, Mosquitto broker allows up to 60,000 messages per second on a machine with a 3 GHz Intel Core 2 Duo processor and 4 GB of RAM.

When it comes to city-wide decision making in *B2S4B* platform, this is a task of the *Analyzer*, that collects high-level events reported by all available *SCOs* published to Kafka broker and thus monitors the whole city. Although decision makers know how many *SCOs* are available in the city, it is not expected that all *SCOs* will run at the same time, and each of them has specific logic in terms of detecting events. The performance and the throughput of data volume sent by *SCOs* to Analyzer could be affected by the person that designs the *SCO* while creating *SCO* definitions. For example, whether one *SCO* detects an event once in 10 minutes or once in an hour highly reduces the overall volume of messages sent to Analyzer via Kafka broker. When it comes to Kafka broker limitations, the broker can handle up to 100,000 messages per second, and could be scaled by adding additional partitions and multiple instances.

Figure 5 shows components of *B2S4B* and data flow, including the distribution of the volume of exchanged messages. As the Figure shows, the amount of messages is reduced exponentially from Things to *SCOs* and then to Analyzer.

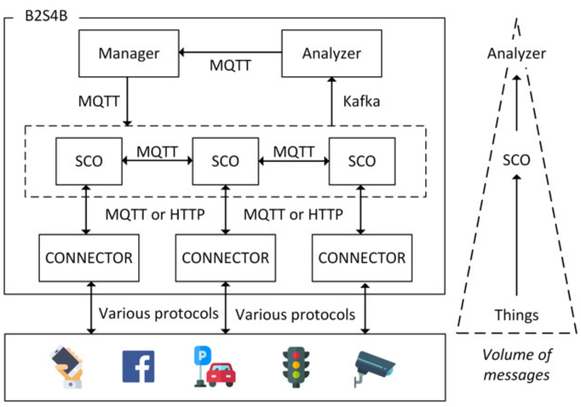

Figure 5. Components of B2S4B and data flow

When it comes to reusability, the *SCO definition* is general by virtue of its structure and could be defined by a domain expert (that does not need to be a technical person) for any other use case, such as smart homes, or environment protection, since there is no need to modify the programming code or the architecture. In that way *SCOs* and the whole platform could be easily deployed and integrated in any BP that runs within the ecosystems of a smart city. Table 1 illustrates the comparison between the B2S4B platform and the existing smart city platforms.

Table 1. Comparison of smart city platforms

	CityPulse	FIWARE	CiDAP	B2S4B
IoT	☑	☑	☑	R
Social networks	☑	☑	R	☑
Modular architecture	☑	☑	-	R
Creates BP	-	-	-	☑
Allows reconfiguration	-	-	-	☑

7. Conclusion

This paper continues the previous efforts for bridging business processes with IoT and Social world into *B2S4B* platform capable to support Smart City so that it could use the full potential of its resources and allow its managers to react fast to unplanned situations.

This paper introduced the concept of *SCO* that observes surroundings, detects events, and performs action(s) when conditions are satisfied. A *SCO* could be a part of a *SCO chain* that represents a BP, and by changing *SCO definition* used for *SCO* initiation, the whole BP is changed.

This paper also presented *Designer*, a user-friendly dashboard that helps any non-technical person to easily create *SCO definitions* and manage existing *SCOs* or deploy new ones. The architecture of a *SCO* is very flexible and allows a high degree of freedom in defining scenarios.

One social connector and one IoT connector were created in order to communicate with both the IoT and social world, but *B2S4B* platform is not limited to those, and could use any 3rd party service as a connector as long as it follows *SCO* guidelines.

By simulating traffic conditions, it was proved that *SCOs* can be chained and used in a real Smart City environment and act as valuable tool for managers and decision makers. Future work will focus on automatic *SCOs* reconfiguration and self-adaption conditions and on solving potential conflicting situations.

As security is concerned, as a big issue and it will also be taken into account in future research. The research agenda would focus on two issues, the first one would be related to vetting the IoT prior to deployment on specific network layer (e.g. fog or cloud), as per the recommendations given in (Masmoudi et. al., 2020), and the second one to securing communications between things, fog and cloud including secure offloading.

Data privacy is an important aspect of this platform, especially when it comes to interacting with Social World. All the data that is considered is publicly available and is the result of public interaction with citizens.

REFERENCES

Anon (n.d.-a) Available at: <http://b2s4b.connect.rs>.

Anon (n.d.-b) Available at: <http://connector.connect.rs>.

Araujo, V., Mitra, K., Saguna, S. & Åhlund, C. (2019). Performance evaluation of FIWARE: A cloud-based IoT platform for smart cities, *Journal of Parallel and Distributed Computing*, *132*, 250-261.

Banks, J. (1999). Introduction to simulation. In *Proceedings of the 31st conference on Winter simulation: Simulation - bridge to the future*, *1*, (pp. 7-13).

Bauman, K., Tuzhilin, A. & Zaczynski, R. (2017). Using social sensors for detecting emergency events: a case of power outages in the electrical utility industry, *ACM Transactions on Management Information Systems (TMIS)*, *8*(2-3), 1-20.

Chamoso, P., González-Briones, A., De La Prieta, F., Venyagamoorthy, G. K. & Corchado, J. M. (2020). Smart city as a distributed platform: Toward a system for citizen-oriented management, *Computer Communications*, *152*, 323-332.

d'Hondt, T., Wilbik, A., Grefen, P., Ludwig, H., Baracaldo, N. & Anwar, A. (2019). Using BPM technology to deploy and manage distributed analytics in collaborative IoT-driven business scenarios. In *Proceedings of the 9th International Conference on the Internet of Things* (pp. 1-8).

DOCKER (n.d.) Available at: <http://docker.com>.

Dumas, M., La Rosa, M., Mendling, J. & Reijers, H. A. (2013). *Fundamentals of Business Process Management*, Springer.

European Commission for Information Society and Media (2008). *Internet of Things in 2020: Roadmap for the Future.* INFSO D.4 Networked Enterprise & RFID INFSO G.2 Micro & Nanosystems in co-operation with the Working Group RFID of the ETP EPoSS.

Faci, N., Maamar, Z., Baker, T., Ugljanin, E. & Sellami, M. (2018). In Situ Mutation for Active Things in the IoT Context. In *13th International Conference on Software Technologies (ICSOFT)*, (pp. 759-766).

Gavrilović, N. & Mishra, A. (2021). Software architecture of the internet of things (IoT) for smart city, healthcare and agriculture: analysis and improvement directions, *Journal of Ambient Intelligence and Humanized Computing*, *12*(1), 1315-1336.

Gharaibeh, A., Salahuddin, M.A., Hussini, S.J., Khreishah, A., Khalil, I., Guizani, M. & Al-Fuqaha, A. (2017). Smart cities: A survey on data management, security, and enabling technologies, *IEEE Communications Surveys & Tutorials*, *19*(4), 2456-2501.

Giatsoglou, M., Chatzakou, D., Gkatziaki, V., Vakali, A. & Anthopoulos, L. (2016). CityPulse: A platform prototype for smart city social data mining, *Journal of the Knowledge Economy*, *7*(2), 344-372

GITHUB (n.d.-a). Available at: <https://github.com/ugljanin/iot-executor>.

GITHUB (n.d.-b). Available at: <https://github.com/CacheControl/json-rules-engine>.

GOJS (n.d.). Available at: <https://gojs.net>.

Janiesch, C., Koschmider, A., Mecella, M., Weber, B., Burattin, A., Di Ciccio, C., Fortino, G., Gal, A., Kannengiesser, U., Leotta, F., Mannhardt, F., Marrella, A., Mendling, J., Oberweis, A., Reichert, M., Rinderle-Ma, S., Serral, E., Song, W., Su, J., Torres, V., Weidlich, M., Weske, M. & Zhang, L. (2020). The Internet of Things Meets Business Process Management: A Manifesto, *IEEE Systems, Man, and Cybernetics Magazine*, *6*(4), 34-44.

Kajan, E., Faci, N., Maamar, Z., Sellami, M., Ugljanin, E., Kheddouci, H., Stojanović, D. H. & Benslimane, D. (2020). Real-time tracking and mining of users' actions over social media, *Computer Science and Information Systems, 17*(2), 403-426.

Kaplan, A. M. & Haenlein, M. (2010). Users of the world, unite! The challenges and opportunities of Social Media, *Business Horizons*, *53*(*1*), 59-68.

Kourtit, K., Nijkamp, P. & Arribas, D. (2012). Smart cities in perspective – a comparative European study by means of self-organizing maps, *Innovation – The European journal of social science research*, *25*(*2*), 229-246.

Maamar, Z., Burégio, V. & Sellami, M. (2015). Collaborative enterprise applications based on business and social artifacts. In *IEEE 24th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises* (pp. 150-155). IEEE.

Maamar, Z., Kajan, E., Guidara, I., Moctar-M'Baba, L. & Sellami, M. (2020). Bridging the gap between business processes and IoT. In *Proceedings of the 24th Symposium on International Database Engineering & Applications* (pp. 1-10).

Masmoudi, F, Zakaria Maamar, Z, Sellami, M., Awad, A.I. & Burégio, V. (2020). A Guiding Framework for Vetting the Internet of Things, *Journal of Information Security and Applications, 55,* 102644.

Mosquitto (n.d.). Available at: <https://www.mosquitto.org/>.

MQTT (n.d.). Available at: <https://mqtt.org>.

Neuman, S. (2015). *Building microservices: Designing fine-grained systems*. O`Reilly & Associates, Inc.

Nigam, A. & Caswell, N. S. (2003). Business artifacts: An approach to operational specification, *IBM Systems Journal*, *42*(3), 428-445.

Portainer (n.d.). Available at: <http://portrainer.io>.

Sánchez, J. R., Campo-Archbold, A., Rozo, A. Z., Díaz-López, D., Pastor-Galindo, J., Mármol, F. G. & Díaz, J. A. (2022). On the power of social networks to analyze threatening trends, *IEEE Internet Computing*, *26*(2), 19-26.

Ugljanin, E., Faci, N., Sellami, M. & Maamar, Z. (2016). Tracking users' actions over social media: Application to Facebook. In *25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE),* (pp. 255-256). IEEE.

Ugljanin, E., Stojanović, D., Kajan, E. & Maamar, Z. (2017). Re-engineering of smart city's business processes based on social networks and Internet of Things, *Facta Universitatis, Series: Automatic Control and Robotics*, *16*(3), 275-288

Ugljanin, E., Kajan, E., Maamar, Z., Asim, M. & Burégio, V. (2020). Immersing citizens and things into smart cities: a social machine-based and data artifact-driven approach, *Computing*, *102*(7), 1567-1586.