Block-based Realtime Big-Data Processing for Smart Cities

Dario Bonino*, Federico Rizzo*, Claudio Pastrone*, José Angel Carvajal Soto†, Matts Ahlsen‡, Mathias Axling‡
*Istituto Superiore Mario Boella, Torino, Italy
{bonino, rizzo, pastrone}@ismb.it
†Fraunhofer FIT, Bonn, Germany
jose.angel.carvajal.soto@fit.fraunhofer.de
‡CNet Svenska AB, Stockholm, Sweden
{matts.ahlsen, mathias.axling}@cnet.se

***Abstract*—Administrators and operators of next generation cities will likely be required to exhibit a good understanding of technical features, data issues, and complex information that, up to few years ago, were quite far from day-to-day administration tasks. In the smart city era, the increased attention to data harvested from the city fosters a more informed approach to city administration, requiring involved operators to drive, direct and orient technological processes in the city, more effectively. Such an increasing need, requires tools and platforms that can easily and effectively be controlled by non-technical people. In this paper, an approach for enabling "easier" composition of real-time data processing pipelines in smart-cities is presented, exploiting a block-based design approach, similar to the one adopted in the Scratch programming language for elementary school students. Language primitives and corresponding REST representations are discussed, showing the viability of the approach. Future works will include experimentation of the proposed concepts in the context of a smart city pilot in Turin, Italy.**

***Keywords*-Complex Event Processing, Block-based programming, Big Data Analysis, Smart City Implementation**

I. Introduction

Cities are playing an increasingly important role, worldwide. They are currently inhabited by nearly half the world's population [1] (68% in Europe[1]), consume 80% of the world's energy production and roughly produce 70% of the total carbon dioxide [2]. Urban-related challenges encompass, for example, (a) the adoption of information and sensing technology for better understanding the city dynamics [3]; (b) the increasing engagement of citizens in administrative and management processes, possibly leveraging the power of the crowds [4], [5]; (c) the environmental sustainability of cities, fostering virtuous behaviors for better consuming energy and goods, and for better handling waste and pollutants [6], [7].

In this extremely complex context, innovative city-wide ICT platforms are deemed as crucial elements for improving city management [8], [9] and, in the long term, city life quality. Several research efforts tackle the issues and challenges [10], [11] related to these "smart cities" [12], e.g., for handling huge amounts of sensors deployed in the city territory [13], [14], to manage administrative processes through open-data exchanges [15], to establish common layers of interoperability between city functions through semantic modeling [16], [17], etc.

Among these challenges, data-handling is particularly interesting. On one hand, it requires technology capable of handling thousands of data points, continuously captured from the field. On the other hand, it requires new programming patterns and paradigms which shall be accessible to user categories that were previously not exposed to hard technological aspects, e.g., city administrators. While many researchers are investigating methods and tools for Complex Event Processing (CEP) [18], [19] and Big Data [20], [21] in the context of smart cities, few approaches explore interaction patterns, programming languages and/or interfaces for enabling non-experts to define on-line data processing pipelines, as it will likely be needed in next generation cities.

In this paper we explicitly target this challenge by proposing a block-based language for defining complex event processing pipelines through composition of simple processing stages, namely blocks. We exploit the proven paradigm of block-based programming, e.g., adopted by the MIT Scratch programming language [22] to enable elementary school students to approach computer science. This paradigm, integrated with the well known pipe and filter pattern [23], permits to design a language composed by simple, easy to understand, processing blocks that composed together can generate complex and articulated processing pipelines. Stemming from previous research on the topic carried by some of the authors [24], [25], in this paper we extend block-based complex event processing to specifically address smart city issues by defining a replicable template primitive, and a wild-card template instantiation algorithm for automatic deployment/duplication of prototypical processing "chains". The enriched set of primitives is given an updated JSON representation and exposed through self-descriptive REST APIs based on the OpenAPI specification[2]. Such APIs, together with their description, might be exploited by web applications to enable easier, graphical creation of stream processing queries, also by non-experts.

The remainder of the paper is organized as follows: Sec-

[1]According to Eurostat, http://ec.europa.eu/eurostat/documents/3217494/5728777/KS-HA-11-001-EN.PDF, last visited on April 27, 2016.

[2]Formerly known as Swagger, available at https://openapis.org, last visited April 21, 2016.

tion II discusses relevant related works highlighting similarities and complementarity with the proposed approach. Section III briefly introduces the basic principles forming the foundations of the proposed approach. More specifically, it provides: (a) an overview of the ALMANAC project from which stems the work presented in this paper; (b) a short summary of the ALMANAC Data Layer where city data is processed and fused to meet particular management needs;(c) a summary of the spChains [25] language extended in this paper. Section IV introduces the block-based Data Fusion Language (DFL) and its smart-city-specific aspects, whereas Section V reports preliminary results with details on corresponding self-describing REST APIs and their implementation. Finally, Section VI finally draws conclusions and proposes future works.

II. Related Works

Complex Event Processing and techniques for handling high-cardinality, high-frequency data streams typical of the Big Data domain have been widely investigated in research. With the advent of Smart Cities, these topics gained an increased momentum related to the inherent challenges brought by the city scenario, from large-scale deployment to interoperability between stream formats, etc. Among the available approaches, a rather evident trend emerges which, on one side recognize that CEP languages are too complex for non-expert users that are required to define processing tasks, and on the other side considers semantic characterization and matching of events a viable solution to address smart city scenarios. Both topics are relevant to the approach proposed in this paper and deserve further investigation.

Anicic et al. [26], define two high-level languages for specifying event patterns named ETALIS Language for Event and Event Processing SPARQL which aim at bridging the gap between domain-expert knowledge and CEP query technicalities. Their approach is similar to the one proposed here as it aims at defining more "effective" and "easy-to-adopt" languages for CEP, with semantic-based definition of events. The proposed DFL moves, in fact, in the same direction and foresees the adoption of SPARQL-based identification of device tuples, in background, to speed-up the process of instantiating CEP pipelines (chains). In the work of Taylor et al. [27] ontologies are used as a basis for the definition of contextualized complex events of interest which are translated to selections and temporal combinations of streamed messages. Supported by description logic reasoning, the event descriptions defined by Taylor et al. are translated to the native language of a commercial CEP engine, and executed under the control of the CEP. This approach shares with the DFL the underlying idea of hiding the complexity of CEP under a more suitable language, exploiting semantics for achieving stream format interoperability and stream matching. However, it targets a rather different community of experts able to successfully understand, and master description logic specification of event processing patterns. The proposed DFL, on the contrary, targets domain experts which are not skilled in CEP but in city administration and organization. As such, its main goal is to trade-off expressiveness and flexibility of CEP languages with easier composition and understandability by non-experts.

Stream matching in large scale scenarios is a widely recognized issue, also shared by the DFL. Hasan et al. in their work [28] examine the requirement of event semantic decoupling and discuss approximate semantic event matching and the consequences it implies for event processing systems. They, in particular define a semantic event matcher and evaluate the suitability of an approximate hybrid matcher based on both thesauri-based and distributional semantics-based similarity and relatedness measures. This approach might be considered for semantic stream matching required by the DFL for wild-card instantiation of chains involving multiple devices.

III. Basic principles

A. The ALMANAC Project

ALMANAC is a EU-funded project which aims at designing and developing a Smart City Platform (SCP) [29] that integrates technologies and paradigms typical of the Internet of Things with edge networks (often named Capillary Networks), by exploiting a cloud-based, federated services approach. The project tackles the full stack of challenges involving smart city platforms, from low-level sensor interfacing and data capture, to high-level support to city processes and policies.

B. Data architecture in ALMANAC

Inside the ALMANAC SCP, data delivery, elaboration and fusion is a core process that enables all supported smart city features. The smart city context, in which the platform is deployed, requires capabilities to handle high-cardinality data streams, with possibly high sampling frequency, depending on enabled processes. For example, in the official project demonstrations the ALMANAC platform handles around 50'000 smart waste bins and over 20'000 water meters, each sampled at intervals between 1 and 10 minutes. Such constraints lead to the development of a dedicated, scalable layer (namely the Data Management Framework) in the platform architecture (see Figure 1), which handles processing of real-time data by exploiting state-of-the-art CEP engines.

The Data Management Framework hosts components for storing city data, for annotating them with respect to well-known ontologies and for applying real-time processing and data fusion. The latter set of functions, in particular, is implemented by the so-called Data Fusion Manager (DFM, Figure 2) which supports processing of live data and events coming from low-level platform components, and delivered through MQTT[3]. The DFM can exploit several event-processing back-ends (e.g., Esper[4], WSO2[5], etc.), each programmed through rather complex, SQL-like, stream processing

[3]OASIS standard, http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf, last visited on April 27, 2016

[4]http://www.espertech.com/esper, last visited on April 27, 2016.

[5]http://wso2.com, last visited on April 27, 2016.

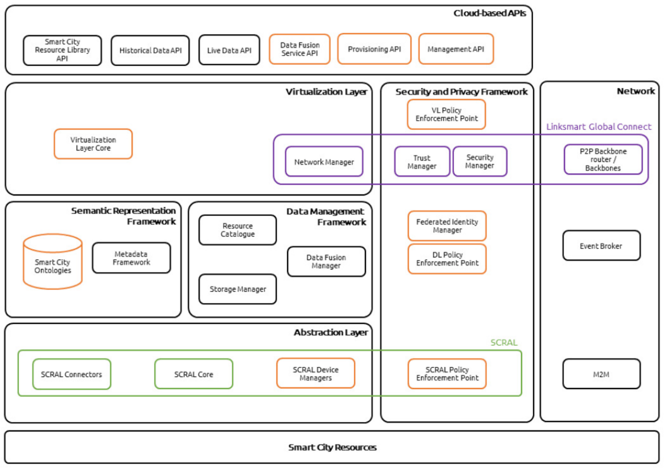


Fig. 1. The ALMANAC platform architecture

languages such as EPL[6], CQL [30] or STREAM [31]. These languages, together with the underlying elaboration engines, enable the platform to sustain high data rates (over 150k events per second on laptop-level machines) while computing complex pattern detection, aggregation and time-wise operations on flowing data streams.

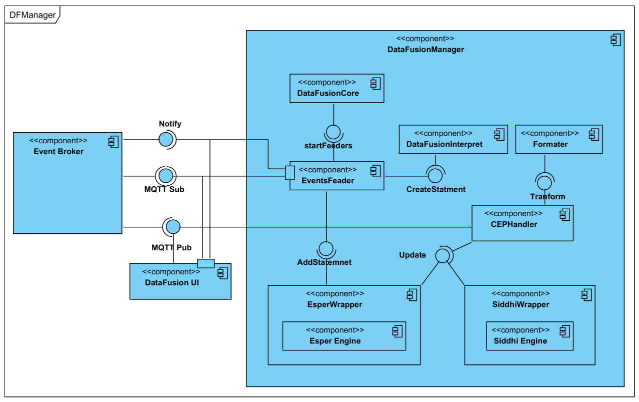


Fig. 2. The ALMANAC Data Fusion Manager

While the DFM supports effective event processing, there is almost no chance for a non-expert to successfully write custom CEP queries for detecting specific patterns or to synthesize particular quantities which are relevant for city operators and/or administrators. To overcome this issue, and bridge the gap between stream elaboration needs, available at the city administration level, and stream processing knowledge, only available at the CEP programming level, we extended and amended the spChains [24] block-based programming paradigm to address city-wide scenarios, thus defining the so-called ALMANAC Data Fusion Language (DFL).

[6]http://www.espertech.com/esper/release-5.4.0/esper-reference/html/index.html, last visited on April 27, 2016

C. *SpChains*

SpChains is a block-based stream processing framework, which represents monitoring (and alerting) tasks as reusable and modular "processing chains" built atop of a set of 14 standard, and extensible, stream processing blocks (Figure 3 reports a sample "smell-detection" chain).

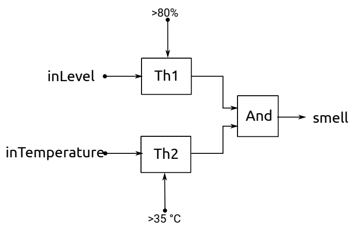
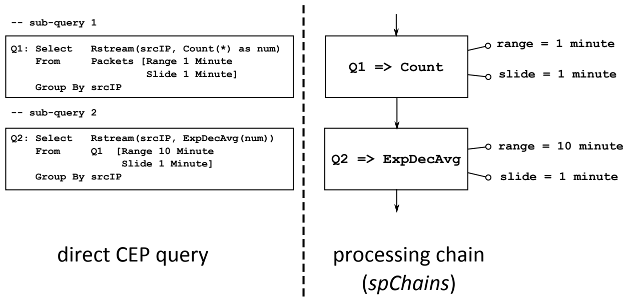


Fig. 3. Bad smell detection

Each block encapsulates a single (parametrized) stream query (see Figure 4), e.g., a windowed average or a threshold check, and can be cascaded to other blocks to obtain complex elaboration chains using a pipes-and-filter composition pattern [23].



```
-- sub-query 1
Q1: Select   Rstream(srcIP, Count(*) as num)
    From     Packets [Range 1 Minute
                      Slide 1 Minute]
    Group By srcIP
-- sub-query 2
Q2: Select   Rstream(srcIP, ExpDecAvg(num))
    From     Q1  [Range 10 Minute
                  Slide 1 Minute]
    Group By srcIP
```

Fig. 4. Direct CEP query mapped on processing blocks

The spChains approach is suitable for small deployments, with few sensors, as it requires to manually define each single processing chain. However, when moving to smart city scenarios, where the number of sensors is huge and possibly unknown, almost impossible to effectively accomplish. In particular, the main obstacles in adopting spChains in a smart city scenario are related to single chain instantiation and to exact, and explicit, source matching (see Figure 5, showing the bad smell chain instantiation). The former requires the chain developer to fully specify chains, even if they apply the same elaboration to different data sources. The latter instead, requires a chain to explicitly and uniquely identify target data-sources, at design time.

IV. Block Based Data Fusion Language

To overcome the spChains issues, we propose the adoption of one new modeling primitive, namely the *template*, and of one new, metadata-based, template instantiation algorithm named *wild-card template binding*.

```
{
  "chains": [
  {
      "id": "bad_smell_1",
      "blocks" : [
          {
            "id" : "Th1",
            "function" :"threshold",
            "params" : [
              {"name" : "threshold", "value" : "80", "uom" : "%"},
              {"name" : "mode", "value" : "rising"}]
          },
          {
            "id" : "Th2",
            "function" :"threshold",
            "params" : [
              {"name" : "threshold", "value" : "35", "uom" : "Celsius"},
              {"name" : "mode", "value" : "rising"}]
          },
          {
            "id" : "And",
            "function" :"and"
          }
        ],
        "connections" : [
           {
            "from" : {"blockId" : "Th1", "ioId" : "out"},
            "to" : {"blockId" : "And", "ioId" : "in1"}
           },
           {
            "from" : {"blockId" : "Th2", "ioId" : "out"},
            "to" : {"blockId" : "And", "ioId" : "in2"}
           }],
        "inputs": [
          {"blockId": "Th1", "port": "in", "ioId": "inLevel"},
          {"blockId": "Th2", "port": "in", "ioId": "inTemperature"}],
        "outputs": [
          {"blockId": "And", "port": "out", "ioId": "smell"}]
    }],
    "bindings":
    [
     {
      "fromSources" : [
        {
          "sourceId" : "WasteBin36754",
          "dataStream" : [
            {"streamId": "Temperature_acd4537fd", "ioId" : "inTemperature"},
            {"streamId" : "FillLevel_cgajh74629", "ioId" : "inLevel"}]
        }
        ],
      "toDrains" : [
        {"drainId": "bad_smell_36754", "ioId" : "smell"}
      ]
    }
  ]
}
```

Fig. 5. Bad smell chain definition and binding to WasteBin36754, in the updated JSON syntax.

A. Template

Single chain instantiation is tackled, for smart city scenarios, by means of the new *template* modeling primitive. Templates are special classes of stream processing chains whose inputs and outputs are left open by exploiting syntactical place-holders, and can be later matched to different sources, and drains (Figure 6).

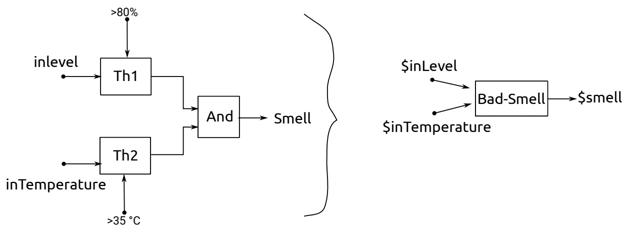


Fig. 6. The *Template* concept.

Templates define “prototypes” of processing chains (which can also be seen as a virtual, complex blocks) that can be instantiated in same way of any other processing block. They ease the processing pipeline instantiation task by reducing the amount of blocks that need to be specifically described in each elaboration chain. The more complex, and the more similar are chains to instantiate, the higher is the gain in terms of time and reduced error rate.

```
{
     "id": "bad_smell_template",
     "blocks" : [
         {
           "id" : "Th1_genid",
           "function" :"threshold",
           "params" : [
             {"name" : "threshold", "value" : "80", "uom" : "%"},
             {"name" : "mode", "value" : "rising"}]
         },
         {
           "id" : "Th2_genid",
           "function" :"threshold",
           "params" : [
             {"name" : "threshold", "value" : "35", "uom" : "Celsius"},
             {"name" : "mode", "value" : "rising"}]
         },
         {
           "id" : "And_genid",
           "function" :"and"
         }
       ],
       "connections" : [
          {
           "from" : {"blockId" : "Th1_genid", "ioId" : "out"},
           "to" : {"blockId" : "And_genid", "ioId" : "in1"}
          },
          {
           "from" : {"blockId" : "Th2_genid", "ioId" : "out"},
           "to" : {"blockId" : "And_genid", "ioId" : "in2"}
          }],
       "inputs": [
         {"blockId": "Th1_genid", "port": "in", "ioId": "$inLevel"},
         {"blockId": "Th2_genid", "port": "in", "ioId": "$inTemperature"}],
       "outputs": [
         {"blockId": "And", "port": "out", "ioId": "$smell"}]
   },
```

Fig. 7. Template syntax example; the “_genid” subfix identifies runtime-generated identifiers whereas “$” identifies free parameters.

For example, Figure 6 and Figure 7 report the bad smell chain rewritten as *template*; free parameters start with a “$” sign. When replicating the template over thousands of bins, the number of required definitions is reduced by a factor of n equal to the number of chain blocks (3 in our case), thus providing a substantial improvement to effectiveness of the language.

B. Wild-card template binding

The term *wild-card template binding*, introduced with the ALMANAC DFL, refers to a more flexible specification of input and output streams for designed templates. While in the original spChains approach (see Figure 5), chain inputs and outputs were tightly bound to uniquely identified data streams, i.e., to specific sources and drains, in the DFL this requirement has been relaxed, and a stream-type matching mechanism is employed to permit *deployment-time* binding of different data streams to the same chain structure. Chains can be defined to be valid for classes of devices of the same kind, generating the same types of data, thus improving the overall instantiation effectiveness.

With wild card binding (syntax exemplified in Figure 8), the chain specification is enriched by:

1) a sourceType “matcher”, which identifies the type of device, e.g., referred to a well known domain ontology, generating the needed data streams.
2) a streamType “matcher”, which identifies the type of stream “suitable” for a given chain input.

```
"bindings":[
  {
    "fromSources" : [{
      "sourceType" : "smartcity:WasteBin",
      "dataStream" : [
      {"streamType" : "smartcity:Temperature", "ioId" : "inTemperature_genid"},
      {"streamType" : "smartcity:FillLevel", "ioId" : "inLevel_genid"}
      ]
     }],
    "toDrains" : [
      {"drainId": "bad_smell_36754", "ioId" : "smell_genid"}
      ]
  }]
```

Fig. 8. *Wild-card template binding* syntax example, only the "bindings" section is affected. The "_genid" subfix identifies runtime-generated identifiers. Source and stream types are referred to the ALMANAC smart city ontology, http://www.almanac-project.eu/ontologies/smartcity.owl.

Currently, the DFL only supports the case of templates attached to a single device instances, which is reasonably straightforward. The corresponding pseudo-code is reported in Algorithm 1.

Algorithm 1 *wild-card template binding* - simple case

```
1: procedure SWC-INSTANTIATION
2:     let $D$ be the set of "selected" devices
3:     for each device $d$ in $D$ do
4:         $ch$ = template.instantiate()
5:         for each $stream$ in device.streams do
6:             /*get the only source available*/
7:             $s$ = getSource($ch$)
8:             /*get the source stream matching the device stream */
9:             $s_{stream}$ = getMatchingSourceStream($stream$,$s$)
10:            attach($s_{stream}$, $stream$)
11:        end for
12:    end for
13: end procedure
```

For templates involving multiple sources, the binding algorithm requires the definition of queries C, possibly matching tuples T of devices. Resulting device tuples are associated to the actual chain inputs by means of the "matcher" fields specified in the *wild-card template binding*. We are still working on the definition of a suitable wild-card binding algorithm for multiple devices. A preliminary, rough version is reported in Algorithm 2 where constraints are matched against device tuples, that can be identified, e.g., by means of SPARQL-based queries sent to the platform device catalog.

Algorithm 2 *wild-card template binding*

```
1: procedure WC-INSTANTIATION
2:     /*constraints on devices*/
3:     let $C$ be ($c_1$, ..., $c_i$)
4:     /*tuples of devices satisfying $C$*/
5:     let $T$ be ($d_{11}$, ..., $d_{1i}$), ..., ($d_{n1}$, ..., $d_{ni}$)
6:     for each tuple $t$ in $T$ do
7:         $ch$= template.instantiate()
8:         for each device $d$ in $t$ do
9:             /*get the source matching the current device*/
10:            $s$ = getMatchingSource($ch$,$d$)
11:            for each $stream$ in $d$.streams do
12:                /*get the source stream matching the device stream */
13:                $s_{stream}$ = getMatchingSourceStream($stream$,$s$)
14:                attach($s_{stream}$, $stream$)
15:            end for
16:        end for
17:    end for
18: end procedure
```

In summary, the process of instantiating same-structured chains for all bins in the city, reduces to a *wild-card template binding* specification integrated by a source search and match process, which can, for example, exploit native features of the smart city platform in which the language is employed.

To better clarify the concept, consider the following smart waste bin example. A sample application of multiple inputs / single output chains in the ALMANAC waste management domain is the bad smell detection chain reported in Figure 3. Defining such a "bad smell" chain for all the city waste bins in a city using spChains, would require a number of full chain specifications equal to the number of bins, e.g., 50,000 in Turin. On the converse, by exploiting the above defined *wild-card template binding*, the same operation will just require one template and binding specification plus a device search on any device repository offered by the smart city platform.

V. Preliminary Results

The proposed block-based Data Fusion Language has been exploited as the high-level CEP definition language for the ALMANAC Data Fusion Manager. A RESTful endpoint has been defined, at the ALMANAC API layer, that enables third party developers to create, update and delete processing chains expressed through the DFL, by using simple HTTP verbs. The endpoint is built on top of a machine-readable API definition compliant with the OpenApi specifications[7], and runs as a Spring-enabled application on a Apache Tomcat servlet container (any servlet container might be used).

Available REST resources, with corresponding HTTP requests, permit to handle the whole life-cycle of chain/template definitions. Table I summarizes the currently available REST resources.

TABLE I
REST API RESOURCES

Resource	**Description**	**Allowed HTTP verbs**
/chains	The collection of all chains registered in the DFM, using the DFL language	GET (read), POST (create)
/chains/{chain-id}	The processing chain identified by the given {chain-id}.	GET (read), PUT (update)
/sources	The sources currently registered in the DFM	GET (read), POST (create)
/sources/{source-id}	The source identified by the given {source-id}	GET (read), PUT (update)
/drains	The drains currently registered in the DFM	GET (read), POST (create)
/drains/{drain-id}	The drain identified by the given {drain-id}	GET (read), PUT (update)
/templates	The templates currently registered in the DFM	GET (read), POST (create)
/templates/{template-id}	The source identified by the given {template-id}	GET (read), PUT (update)

[7]https://openapis.org/specification, last visited on April 27, 2014.

At the time of writing, the DFL REST API is part of the official ALMANAC platform instances in Turin, both for production and test virtual machines adopted by the project[8]. It has been continuously working since the last 6 months with no down times and/or major failures.

VI. Conclusions and Future Works

In this paper we presented the ALMANAC DFL: an extension of previous works on block-based CEP, in the context of smart cities. Two main issues of former approaches, i.e., single chain instantiation and direct source binding, have been tackled through a new *template* primitive and an advanced template instantiation algorithm, named *wild-card template binding*. Future works will include further, in deep, investigation of the proposed algorithms for chains involving multiple devices, e.g., by exploiting semantic-based matching of device tuples and chain inputs. The experimentation of the proposed concepts will be carried in the context of the ALMANAC smart city pilot in Turin, Italy.

VII. Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n.609081.

References

[1] "State of world population 2007, unleashing the potential of urban growth," U.N. Population Fund (UNFPA), Tech. Rep. [Online]. Available: http://www.unfpa.org/public/publi-cations/pid/408

[2] "Sustainable smart cities - building sustainable business value in changing cities," KPMG Asset Management Competence Centre, Tech. Rep., 2012.

[3] R. Petrolo, V. Loscr, and N. Mitton, "Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms," *TRANSACTIONS ON EMERGING TELECOMMUNICATIONS TECHNOLOGIES*, 2015.

[4] D. Doran, S. Gokhale, and A. Dagnino, "Human sensing for smart cities," in *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*, Aug 2013, pp. 1323–1330.

[5] J. Gabrys, "Programming environments: Environmentality and citizen sensing in the smart city," *Environment and Planning D: Society and Space*, vol. 32, no. 1, pp. 30–48, 2014.

[6] A. Medvedev, P. Fedchenkov, A. Zaslavsky, T. Anagnostopoulos, and S. Khoruzhnikov, *Internet of Things, Smart Spaces, and Next Generation Networks and Systems: 15th International Conference, NEW2AN 2015, and 8th Conference, ruSMART 2015, St. Petersburg, Russia, August 26-28, 2015, Proceedings*. Cham: Springer International Publishing, 2015, ch. Waste Management as an IoT-Enabled Service in Smart Cities, pp. 104–115. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-23126-6_10

[7] L. Snchez, I. Elicegui, J. Cuesta, and L. Muoz, "On the energy savings achieved through an internet of things enabled smart city trial," in *2014 IEEE International Conference on Communications (ICC)*, June 2014, pp. 3836–3841.

[8] I. Vilajosana, J. Llosa, B. Martinez, M. Domingo-Prieto, A. Angles, and X. Vilajosana, "Bootstrapping smart cities through a self-sustainable model based on big data flows," *Communications Magazine, IEEE*, vol. 51, no. 6, p. 128134, June 2013.

[9] D. Bartlett, W. Harthoorn, J. Hogan, M. Kehoe, and R. J. Schloss, "Enabling integrated city operations," *IBM Journal of Research and Development*, vol. 55, no. 1.2, pp. 15:1–15:10, Jan 2011.

[10] A. Ojo, Z. Dzhusupova, and E. Curry, "Exploting the nature of the smart cities research landscape," *Public Administration and Information Technology*, vol. 11, 2016.

[11] M. Batty, K. Axhausen, F. Giannotti, A. Pozdnoukhov, A. Bazzani, M. Wachowicz, G. Ouzounis, and Y. Portugali, "Smart cities of the future," UCL Centre for Advanced Spatial Analysis, Tech. Rep., October 2012.

[12] "Smart cities. intelligent information and communications technology infrastructure in the government, buildings, transport, and utility domains." Pike Research, Tech. Rep., 2011. [Online]. Available: http://www.navigantresearch.com/research/smart-cities

[13] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, Feb 2014.

[14] S. K. Datta and C. Bonnet, "Internet of things and m2m communications as enablers of smart city initiatives," in *Next Generation Mobile Applications, Services and Technologies, 2015 9th International Conference on*, Sept 2015, pp. 393–398.

[15] A. Cenedese, A. Zanella, L. Vangelista, and M. Zorzi, "Padova smart city: An urban internet of things experimentation," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, June 2014, pp. 1–6.

[16] M. D'Aquin, J. Davies, and E. Motta, "Smart cities' data: Challenges and opportunities for semantic technologies," *IEEE Internet Computing,*, vol. 19, no. 6, pp. 66–70, Nov-Dec 2015.

[17] N. Komninos, C. Bratsas, C. Kakderi, and P. Tsarchopoulos, "Smart city ontologies: Improving the effectiveness of smart city applications," *Journal of Smart Cities*, vol. 1, September 2015.

[18] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[19] O. Etzion and P. Niblett, *Event Processing In Action*. Manning Publications and co., 2010.

[20] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs, "Building a big data platform for smart cities: Experience and lessons from santander," in *2015 IEEE International Congress on Big Data*, June 2015, pp. 592–599.

[21] Z. Khan, A. Anjum, and S. L. Kiani, "Cloud based big data analytics for smart future cities," in *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on*, Dec 2013, pp. 381–386.

[22] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *ACM Transactions on Computing Education*, vol. 10, no. 4, pp. 16:1–16:15, Nov. 2010. [Online]. Available: http://doi.acm.org/10.1145/1868358.1868363

[23] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-oriented Software Architecture Volume 1*. John Wiley & Sons, 1996.

[24] D. Bonino, F. Corno, and L. De Russis, "Real-time big data processing for domain experts, an application to smart buildings," in *Big Data Computing / Rajendra Akerkar*. Boca Raton: CRC Press, 2013, vol. 1, pp. 415–447.

[25] D. Bonino and L. De Russis, "Mastering real-time big data with stream processing chains," *CROSSROADS*, vol. 19, no. 1, pp. 83–86, 2012.

[26] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic, "Stream reasoning and complex event processing in etalis," *Semantic Web*, vol. 3, no. 4, pp. 397–407, 2012.

[27] K. Taylor and L. Leidinger, "Ontology-driven complex event processing in heterogeneous sensor networks," in *Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web: Research and Applications - Volume Part II*, ser. ESWC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 285–299.

[28] S. Hasan, S. O'Riain, and E. Curry, "Approximate semantic matching of heterogeneous events," in *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '12. New York, NY, USA: ACM, 2012, pp. 252–263. [Online]. Available: http://doi.acm.org/10.1145/2335484.2335512

[29] D. Bonino, M. T. D. Alizo, A. Alapetite, T. Gilbert, M. Axling, H. Udsen, J. A. C. Soto, and M. Spirito, "Almanac: Internet of things for smart cities," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, Aug 2015, pp. 309–316.

[30] S. Traut and P. Purich, "Cql language reference for oracle complex event processing," Oracle, Tech. Rep., 2012.

[31] T. S. Group., "STREAM: The Stanford Stream Data Manager," Stanford InfoLab, Tech. Rep., 2003.

[8]Ubuntu 14.04LTS, 6GByte RAM and Intel Xeon processor.