A Self-healing Framework for Online Sensor Data

Tuan Anh Nguyen and Marco Aiello
University of Groningen
Groningen, The Netherlands
Email: {t.a.nguyen, m.aiello}@rug.nl

Takuro Yonezawa
Keio University
Tokyo, Japan
Email: takuro@ht.sfc.keio.ac.jp

Kenji Tei
National Institute of Informatics
Tokyo, Japan
Email: tei@nii.ac.jp

***Abstract*—In pervasive computing environments, wireless sensor networks (WSNs) play an important role, collecting reliable and accurate context information so that applications are able to provide services to users on demand. In such environments, sensors should be self-adaptive by taking correct decisions based on sensed data in real-time. However, sensor data is often faulty. Faults are not so exceptional and in most deployments tend to occur frequently. Therefore, the capability of self-healing is important to ensure higher levels of reliability and availability. We design a framework which provides self-healing capabilities, enabling a flexible choice of components for detection, classification, and correction of faults at runtime. Within our framework, a variety of fault detection and classification algorithms can be applied, depending on the characteristics of the sensor data types as well as the topology of the sensor networks. A set of mechanisms for each and every step of the self-healing framework, covering detection, classification, and correction of faults are proposed. To validate the applicability, we illustrate a case study where our solution is implemented as an adaptation engine and integrated seamlessly into the ClouT system. The engine processes data coming from physical sensors deployed in Santander, Spain, providing corrected sensor data to other smart city applications developed in the ClouT project.**

I. Introduction

Wireless sensor networks (WSNs) have found their way into a wide variety of applications and systems, particularly in pervasive computing environments. Smart cities are an example where WSNs provide the basic infrastructure for data collection and the creation of context-aware systems. Imagine a city filled with various sensors for traffic, pollution, weather, etc., anything is possible. At each point in time, these sensors generate data specific to that city at that moment. Now imagine that you can retrieve and sort that data flow in real-time to offer it to various applications. This raw data can then been combined to offer high level information that can in turn be used by various tailored, responsive services designed to improve quality of life for the city residents. In such smart environments, WSNs are expected to be self-managing systems since self-management is an effective approach for WSNs to tackle the increasing complexity of managing modern-day smart environments, and empower the WSNs to reconfigure themselves to deal with internal and external dynamics in their environment without human intervention [17].

Self-healing has been widely considered as a key property of self-management systems. Self-healing refers to an ability of the systems to "heal" themselves in the sense of recovering from faults and regaining normative performance levels independently. The concept derives from a manner in which a biological system heals a wound [9]. Recent growth of interests in WSNs has strengthened the importance of self-healing [2]. The reason is that faults in WSNs are not exceptional and tend to occur more frequently in most deployments. In addition to typical network faults, data from WSNs suffer from faults arising out of unreliable hardware, limited energy, connectivity interruption, etc. Thus, to ensure the reliability and availability of smart environment applications, self-healing is essential to monitor and analyse sensor data coming from WSNs in order to autonomously examine, diagnose, and react to faults.

We propose a framework that provides runtime self-healing capabilities, enabling a flexible choice of components for fault detection, classification, and correction. The framework provides self-healing as a service to other systems, to compose the self-healing service systematically and consistently, being able to perform self-healing at runtime. The framework also permits flexibility in more easily modifying the self-healing service over time rather than reimplementing the system.

The major contributions of our work are: 1) a general and flexible framework for self-healing of online sensor data; 2) a set of mechanisms for each step of the self-healing framework, covering fault detection, classification, and correction; and 3) a case study that shows how the proposed solution is appropriate and can be implemented in a real-world system, the ClouT framework [16], in order to support smart city applications.

In the following, we present the background of self-healing process as one major property of self-adaptive systems (Section II). Section III describes the design of our proposed self-healing framework together with appropriate resilience mechanisms for each process of the framework, followed by Section IV that presents the implementation of our framework in the ClouT project. Finally, Section V discusses the potential of our proposed framework and suggest future work.

II. Self-Healing Process and Self-Adaptation

The term "self-healing" is drawn from the natural biological paradigm. Through billions of years nature has created extraordinary mechanisms to achieve robustness and enabled self-healing [9]. The main purpose of self-healing is that a system maintains itself at its normality or tries to recover from a degraded state. Thus, before a system adjusts itself, it needs to be cognisant of its *"normal"*, *"degraded"* and *"broken"* states. A system operates healthily at its normal state. A system usually transits from a normal state to a degraded state as faults begin to take effect. At a degraded state, the system would perform some mechanism to heal the faults to transit back to the normal state. A system transits from a degraded state to a broken one when it cannot heal the faults. External interventions are required to help the system recover back to its normal state.

A. The Self-healing States and Model

Inspired by the self-healing process of natural systems discussed in [9], we propose a self-healing model for sensor data, Figure 1, in which each sensor data node could be at one of three states, namely *normal*, *faulty*, and *failure*. A node transits from one state to another depending on its current sensor reading and its capabilities of fault healing. At the *normal state*, data received from sensor are monitored and analysed to identify and detect faulty readings. Once faults are detected the sensor data node should change its state to *faulty state*. At the faulty state, classification functions should diagnose the faults, classify them and apply some correction mechanisms to correct faults. In case the system is able to heal the faulty data, the state of the sensor data node changes back to normal and the node continues to monitor its data. If the faults are not healed, the data node should change its state to broken, notifying other calling services and administrators to take necessary actions in order to bring the sensor node back to normal state, e.g. fixing the physical sensors.

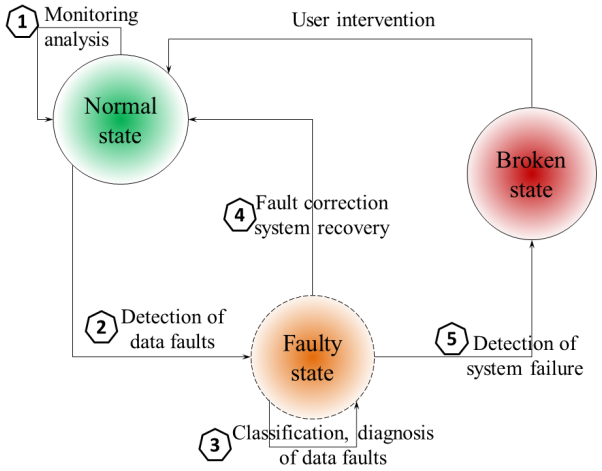

Fig. 1. State diagram of self-healing and its transition functions

As next steps, we propose mechanisms to automatise processes of the self-healing model, realising them in a self-healing framework for sensor data. Our framework is based on and greatly supported by a well-known model for self-adaptation systems, the MAPE-K model [7]. The automatised processes, see Figure 1, are (1) *monitoring and analysis of sensor data*, (2) *detection of data faults*, (3) *classification and diagnosis of data faults*, (4) *fault correction and system recovery*, and (5) *detection of system failure and user notification*.

B. The MAPE-K Architecture for Self-Adaptation

Self-healing is one of four main properties of a self-adaptive system [14]. Self-healing is a specific case of a self-adaptive system, that particularly focuses on realising the self-healing property. For self-adaptive system realisation, IBM introduced the MAPE-K control loop mechanism [7]. The MAPE-K is later discussed in the context of self adaptive systems and is referred to as the adaptation loop. The MAPE-K adaptation loop, Figure 2, includes *Monitor*, *Analyse*, *Analyse*, *Plan* processes, and a shared *Knowledge Base*.

By definition, the monitor process collects the details, such as topology information, configuration property settings from managed resources. The Monitor process aggregates and filters these details until it determines a symptom that needs to be analysed. The Analyse process performs data analysis and reasoning on the symptoms provided by the Monitor process. The Analyse process is influenced by stored knowledge data. If changes are required, a change request is passed to the Plan process. The Plan process structures actions needed to achieve goals. The Plan process creates or selects a procedure to enact a desired alteration in the managed resources. The Plan process can take on many forms, ranging from a single command to a complex workflow. The Execute process changes the behaviour of the managed resource based on the actions recommended by the Plan process. All four processes share a knowledge base that includes data, such as historical logs, symptoms, or policies. The knowledge base might be updated by the processes depending on the process outcomes.

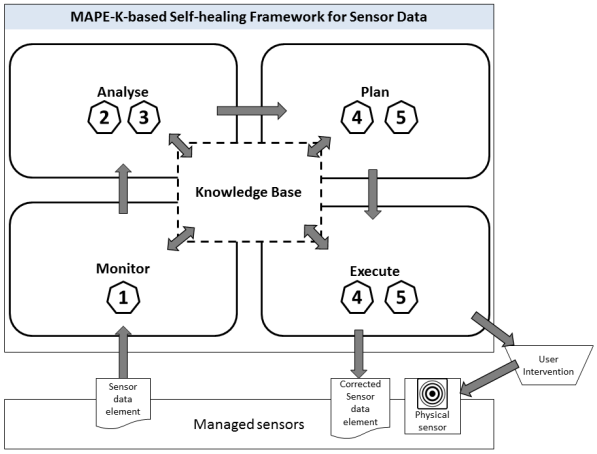

Fig. 2. The MAPE-K architecture for Self-adaptation systems

Many recent studies have investigated MAPE-K model for self-management of WSNs as autonomic computing systems, addressing different self-* properties. However, most of the studies target self-protection, self-optimisation, or self-configuration properties [18]. Only few of the recent studies try to apply MAPE-K for self-healing property [12]. Additionally, those studies apply the model at different network protocol layers of WSNs. For example, [5] tries to reconfigure nodes in order to deal with faulty physical sensors while in [10] the authors propose a self-healing framework to handle low bit error rate and packet lost ratio at the network layer of WSNs. The Agilla [8] provides a programming model with which applications can self-heal by cloning or moving their agent onto the replacement node.

We distinguish our work by proposing a MAPE-K based framework for self-healing of online sensor data. We employ a data-centric, diagnostic approach that looks at faults without considering the physical reasons underneath. The data-centric approach describes the faults by their patterns and behaviours. In this way, we can remove the dependencies on network or hardware causes of data faults. Moreover, application layer approach is efficient, and it can address faults in any type of resources. We believe that our proposed fault detection, classification, and correction mechanisms provide ways to combine benefits of the centralised approach, e.g. computa-

tional power with the flexibility of in-node correction. The network would be able to recognise faults on each node, handle it appropriately and keep the communication overhead low.

In our proposed framework based on MAPE-K model, we realise the (1) *monitoring and analysis* as a function of the Monitor process, the (2) *detection of data faults* and the (3) *classification and diagnosis of data faults* as functions of the Analyse process, while the Plan and Execute processes realise the (4) *fault correction and system recovery* and the (5) *detection of system failure and user notification*. The details of our proposed MAPE-K based self-healing framework for sensor data are discussed in the following section.

III. A MAPE-K-based Self-healing Framework for Online Sensor Data

According to the aforementioned self-healing model, our proposed framework would be equipped with necessary mechanisms for each state of a sensor data element. Our ultimate idea is to provide a flexible framework with which suitable mechanisms can be used in different processes of the framework. Nevertheless, in this paper, we propose some mechanisms for each of the processes. The proposed mechanisms provide the runtime capabilities for detection, classification, and correction of faults that appear in sensor data.

The functionalities of the Monitor and the Analyse processes can be overlapped. For example a fault, i.e. symptom, can be detected by either the Monitor as suggested in the original MAPE-K [7] or by the Analyse as suggested in a revised MAPE-K [6]. In our framework, for the consistency of the proposed fault handling mechanisms, the Analyse takes care of both fault detection and classification. Figure 3 illustrates the framework in more details. This framework addresses the full cycle of the self-healing model, which includes 1) monitoring and analysis as well as fault detection at normal state; 2) diagnosis and classification of faults at faulty state; 3) resiliency and fault correction mechanisms to help system recover to normal state from a faulty one supported by a complete and consistent fault model, and 4) sensor data faults can not be healed, fault notification transits the data node to broken state, notifying other calling services and system administrators to take necessary actions against detected faults and to bring the sensor node back to normal state.

A. Fault Classification Model

Since the main purpose of the self-healing framework is to manage sensor data faults, it is crucial to categorise faults. By comprehending the causes, effects, and especially the characteristics of each fault type, it is possible to propose suitable fault-tolerance mechanisms to detect, classify, and correct faults of each type. Especially in the absence of ground truth, data modelling is vital. Therefore, before discussing the processes, we present the Baljak [3] fault model used in our self-healing framework. The Baljak fault model is based on the **the frequency and continuity of fault occurrence** and on **observable and learnable patterns** that faults leave on the data. This categorisation is flexible and applicable to a wide range of sensor readings. The underlying cause of the error does not affect this categorisation, which makes it possible to handle the faults based on their patterns of occurrence on each sensor node. The Baljak model provides a decision tree to classify data faults into four types: 1) *bias fault*, 2) *drift fault*, 3) *malfunction fault*, and 4) *random fault*.

- **Intermittent** – Faults occur from time to time, and the occurrence of faults is discrete.
  - **Malfunction** – Faulty readings appear frequently. The frequency of the occurrences of faults is higher than a threshold τ.
  - **Random** – Faults appear randomly. The frequency of the occurrences of faulty readings is smaller than τ.
- **Regular** – During the period under observation, faults occurs constantly, and it is possible to observe a pattern in the form of a function.
  - **Bias** – The function of the error is a constant. This can be a positive or a negative offset.
  - **Drift** – The deviation of data follows a learnable function, such as a polynomial change.

B. Knowledge Base Initialisation at the Calibration Phase

A system is usually calibrated before being deployed in a real environment. At the calibration phase, necessary knowledge and assumptions about the environment as well as the system are gathered in order to build the knowledge base that is used later for the sake of self-healing. The knowledge base includes 1) the information about environment, 2) the database of managed sensor data elements, 3) models for fault detection and classification used at the Analyse process, 4) models for fault correction used by the Plan process, and other necessary information. One should notice that the knowledge base should be updated by the processes automatically and by administrators manually to keep it up to date.

C. Sensor Data Monitoring at the Monitor Process

At the early state of system deployment, right after the calibration phase, all sensor elements are at their *normal state*, i.e. there is no faults in sensor readings. Based on this assumption, the responsibility of the Monitor process is to 1) collect real-time readings from managed physical sensors, 2) retrieve necessary historical data for all sensors using pre-defined parameters, 3) gather real-time readings from neighbourhood sensor elements defined accordingly to a neighbouring model, 4) annotate the state of each and every sensor reading based on some assumption and information. One should notice that models, assumptions used are retrieved from the Knowledge Base. After this pre-processing step, necessary sensor data are passed to the Analyse process for sensor data diagnosis.

D. Fault Detection and Classification at the Analyse Process

Sensor data received from the Monitor process are analysed actively to identify and detect faulty readings in sensor data. A variety of fault detection and classification algorithms can be applied at this state, depending on the characteristics of sensor data as well as the topology of the sensor networks. In this paper, we apply a hybrid mechanism for fault detection using neighbourhood vote together with time series data analysis proposed in [11]. Meanwhile the fault classification implements Baljak fault model, discussed in Section III-A.

At the detection phase, each sensor data element compares its current reading with 1) the value computed by the neighbourhood voting technique, and 2) the value forecast by

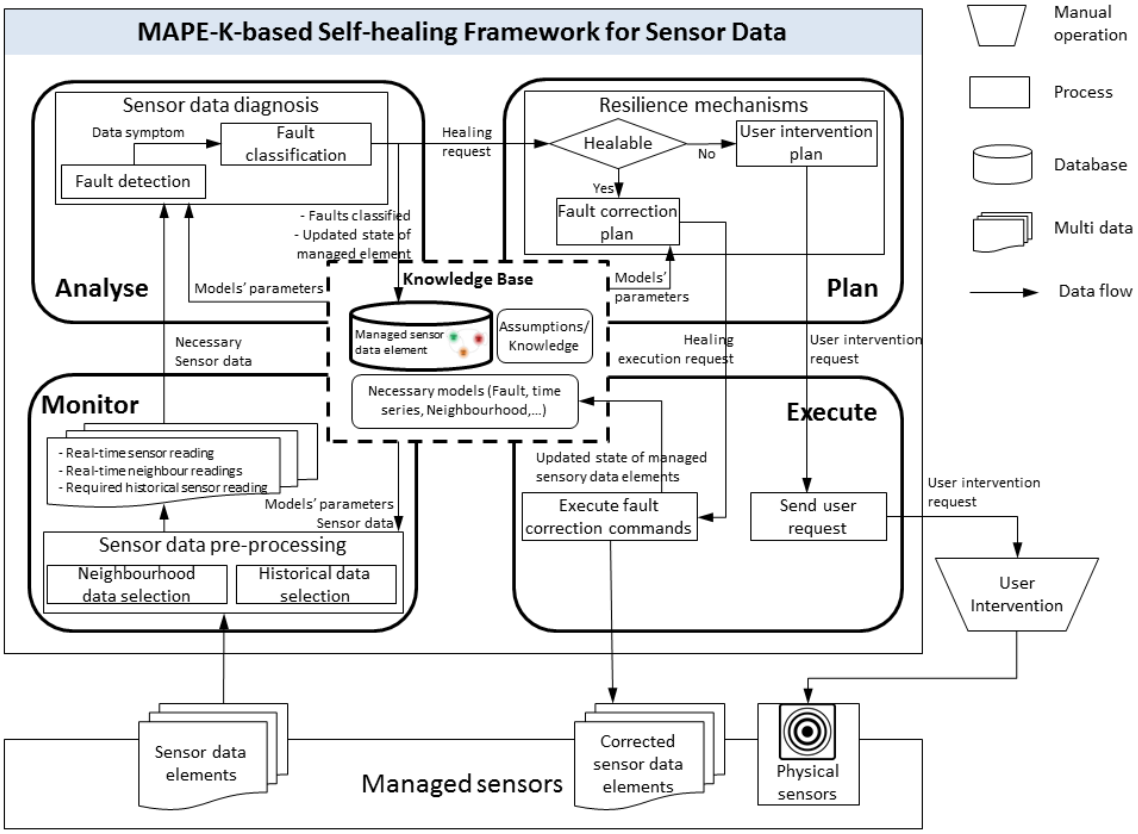

Fig. 3. The MAPE-K-based framework for self-healing of sensor data

the series data forecasting model. The result of the detection phase is the state of the reading examined. In case a sensor reading is detected as faulty, i.e. a symptom, it is diagnosed by the fault classification component, which is a part of the diagnosis procedure. Details of the fault detection and classification mechanisms are thoughtfully discussed in [11]. The fault model use for classification is stored in the shared Knowledge Base. The model is also applied later at the Plan process to correct readings from the respective faulty nodes. Moreover, the framework flexibly allows different set of applicable algorithms to be implemented at each process.

After diagnosing, the Analyse process updates the faulty state of managed sensor data elements back to the shared database in the Knowledge Base. In this way, the state of all managed sensor data elements are synced and consistent among components of the framework. As an outcome, the Analyse process sends healing requests to the Plan process, providing a list of faulty sensor data elements that require the Plan process to take necessary actions, either heal if possible or notify users to take particular interventions.

E. Proposed Fault Correction Mechanisms at the Plan Process

The Plan process structures actions needed to achieve goals. In this case, the goals are to correct the faulty sensor readings if possible, maintaining the managed elements at their normal state as well as providing corrected sensor data to external services that are consuming the data. In case the faults can not be healed, the Plan process should notify external services and system administrators to take appropriate actions, e.g. fixing physical sensors, in order to bring the managed sensor data elements back to their normal state again. For the sake of fault correction, we believe that the complete and consistent fault model used makes it possible to comprehend the causes, effects, and the characteristics of each fault type. Therefore, suitable mechanisms are proposed to correct faults of each type. The *resilience mechanisms* decide how to handle identified data faults. In particular,

- For random and malfunction faults, voting techniques among neighbours are widely used to replace faulty readings with the ones from healthy neighbours. In addition, random faults could also be replaced by interpolating between the two healthy readings before and after the faulty interval.
- Bias and drift faults are more interesting and difficult to correct. Model learning would address the issue. Since faults only belong to a limited number of proposed classes, statistical pattern recognition is suitable to learn appropriate fault models. A fault model for each faulty node can be learned from 1) expectations of correct behaviour established at the calibration phase and 2) the historical sensor data. For example, in [15] the authors consider a low order polynomial model for drift faults, whereas bias are presented by a constant offset. Model learning for fault correction is in the scope of our future work.

In case faults cannot be healed, The *user intervention plan* notifies external calling services and system administrators are informed to take appropriate actions, e.g. fixing physical

sensor, to bring the managed sensor data elements back to their normal state. Additionally, the system could offer readings from neighbours as alternatives to external calling services.

F. The Execute Process

The Execute process executes fault correction commands or sends user requests, depending on the instructions received from the Plan. The former function correction commands in order to fix the faulty sensor data elements and update their state to the Knowledge Base as well as to provide corrected sensor data elements to external calling services, while the latter function is to inform system users or administrators to take necessary interventions, such as fixing physical sensors.

IV. The ClouT Case Study

The ClouT [16] is a collaborative project between Europe and Japan. Its overall concept is leveraging cloud computing as an enabler to bridge the Internet of Things with Internet of People via Internet of Services, establishing an efficient communication and collaboration platform. The main goal of ClouT is to design, implement, and validate a reference IoT+cloud architecture for a smart city ecosystem that helps city authorities provide the backbone for the innovation of their environments.

Figure 4 shows the architectural layers of the city infrastructure, based on the typical cloud stratification. At the bottom the *City Infrastructure as a Service (CIaaS)* is a set of interconnected physical resources all made homogeneous using virtualisation technologies. The CIaaS exposes virtualised resources by mean of a set of open standard APIs that enable the seamless access to any device, data, and computational power. The *City Platform as a Service (CPaaS)* layer consists of a set of specialised middleware services that enable specifically the mash-up and development of applications for the citizens (end users) and the city managers (administrators). The *City Software as a Service (CSaaS)* layer represents a number of applications developed using the CPaaS and CIaaS APIs and running over the cloud on onboard of mobile devices.

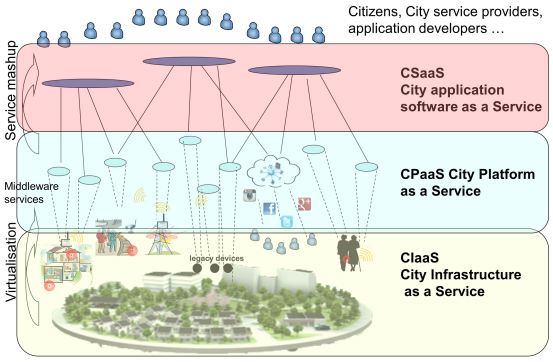

Fig. 4. Cloud model for IoT Services

A. Self-healing as a Service of CPaaS Layer

More specifically, the CPaaS layer, Figure 5, includes the development and processing tools offered by the ClouT platform. The main function of this layer is to build the data processing features of the ClouT platform. It gathers city data via the City Resource Access component and provides processed data/events, as well as high level contextual information to the applications or to the service composition environment. In the CPaaS layer, the *City Data Processing*

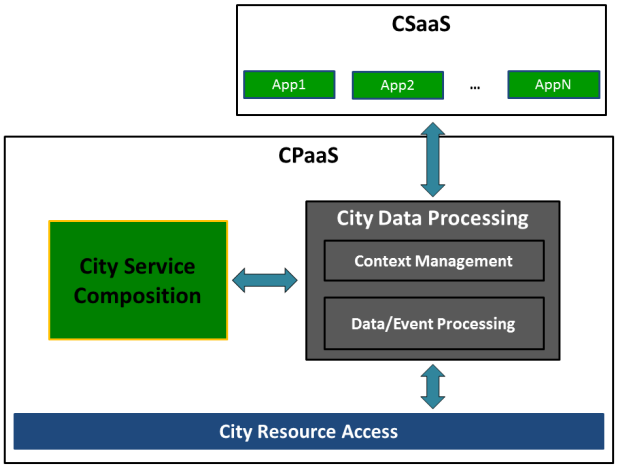

Fig. 5. Main functional blocks of the CPaaS and the interactions between the City Data Process and other blocks

component enables the analysis and extraction of the data archived inside the cloud storage. The City Data Processing offers querying interfaces that are exposed to the overlaying services. The component includes an event/decision making engine that analyses requested queries and takes actions based on a configurable rules directory. The City Data Processing is also responsible for data fault analysis and rectification of faulty sensor data.

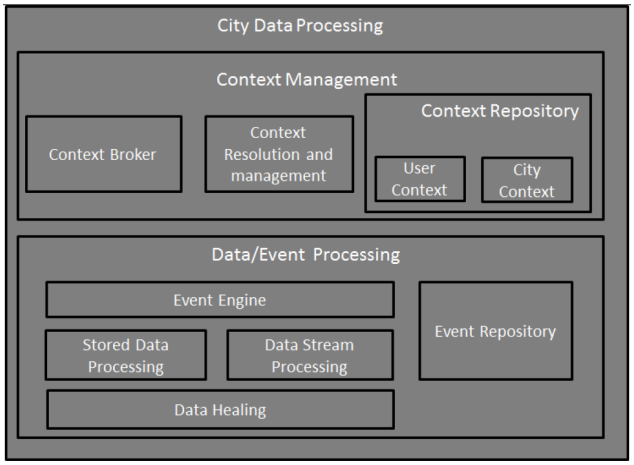

Fig. 6. The City Data Processing architecture

The city data processing component is composed of two main functional blocks: 1) *Data/event processing* in charge of handling data collected by various city data and event sources, and 2) *Context management* in charge of storing and delivering high level context information obtained from processed data and events. The interactions between the City Data Processing block and other blocks in the system are also illustrated in Figure 5, while Figure 6 provides a closer look at

the components of the City Data Processing block. As can be seen, the *Data Healing* component is responsible of identifying and correcting faults in stored data as well as sensor data streams. We represent an implementation of the Data Healing component by using our proposed self-healing framework.

B. Self-healing Framework for the ClouT Platform

The ClouT Data Healing service realises our framework as its proactive *Adaptation Engine* that gathers, via the *Data Exchanger*, online sensor data from the City Resource Access components and applies our self-healing framework to correct faults in the data. The component is invoked by external services that require self-healing service though the *Data Exchanger* interface, Figure 7.

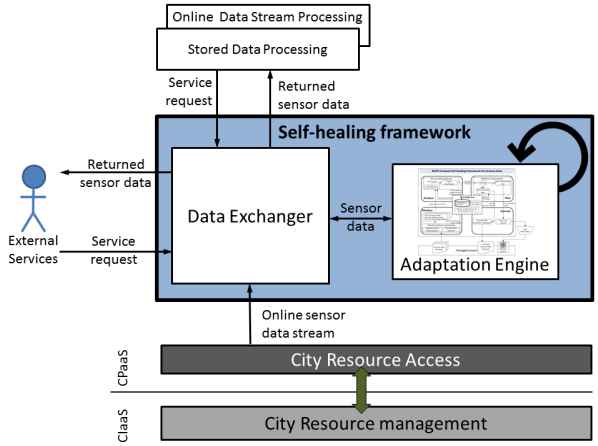

Fig. 7. General architecture of the ClouT self-healing framework

The implementation of our framework is integrated seamlessly into the ClouT system thanks to the Keio virtual sensor system [1], that is based on Sensor-Over-XMPP [4] originally developed by the Sensor Andrew Project [13]. By using the Keio system, the Data Exchanger component is able to create virtual sensor data elements associated with physical sensors deployed in the environment. The virtual elements subscribe to the physical sensors to retrieve actual readings at runtime and pass the data via the Data Exchanger to the Adaptation Engine for processing. Corrected sensor data are then sent back to the Data Exchanger for publishing to the virtual sensor data elements. One can refer to [1] for detailed implementation and user manual of the Keio system as well as how our self-healing framework is integrated into the ClouT platform.

V. Discussion and Future Work

In this paper, we propose a framework that is capable of providing runtime self-healing service to other systems and users. The framework enables a flexible choice of components and mechanisms for fault detection, classification, and correction. We not only present a high level architecture of our self-healing framework but also an implementation of our proposed framework. More specifically, we show how our framework is seamlessly integrated and used in a real-world system, the ClouT platform for smart city applications. Our future work will focus on the improvements of mechanisms for fault detection and classification, especially for fault correction. We also plan to focus on evaluating our solution on top of an operational platform for smart environments, such as ClouT, in order to validate the availability and reliability of applications.

Acknowledgement

The work is supported by 1) the FP7-ICT-2013-EU-Japan, Collaborative project ClouT, EU FP7 Grant number 608641; NICT management number 167 and 2) the Dutch National Research Council Energy Smart Offices project, contract no. 647.000.004.

References

[1] The Keio Virtual Sensor System for ClouT: http://sox.ht.sfc.keio.ac.jp/.

[2] M. Asim, H. Mokhtar, and M. Merabti. A self-managing fault management mechanism for wireless sensor networks. *arXiv preprint arXiv:1011.5072*, 2010.

[3] V. Baljak, T. Kenji, and S. Honiden. Faults in Sensory Readings: Classification and Model Learning. *Sensors & Transducers*, 18:177–187, 2013.

[4] G. Bhatia, A. Rowe, M. Berges, and C. Spirakis. Sensor-Over-XMPP: http://www.xmpp.org/extensions/inbox/sensors.html. 2011.

[5] T. Bourdenas and M. Sloman. Starfish: policy driven self-management in wireless sensor networks. In *Proc. of the ICSE Workshop on Soft. Eng. for Adaptive and Self-Managing Sys.*, pages 75–83. ACM, 2010.

[6] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, and M. Shaw. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, pages 48–70. Springer, 2009.

[7] A. Computing et al. An architectural blueprint for autonomic computing. *IBM White Paper*, 2006.

[8] C.-L. Fok, G.-C. Roman, and C. Lu. Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(3):16, 2009.

[9] D. Ghosh, R. Sharman, H. R. Rao, and S. Upadhyaya. Self-healing systemssurvey and synthesis. *Decision Support Systems*, 42(4):2164–2185, 2007.

[10] R. Muraleedharan and L. A. Osadciw. Secure self-adaptive framework for distributed smart home sensor network. In *Signals, Systems and Computers, 2009 Conference Record of the Forty-Third Asilomar Conference on*, pages 284–287. IEEE, 2009.

[11] T. A. Nguyen, D. Bucur, M. Aiello, and K. Tei. Applying time series analysis and neighbourhood voting in a decentralised approach for fault detection and classification in wsns. In *Proc. of the 4th Symposium on Information and Communication Tech.*, pages 234–241. ACM, 2013.

[12] J. M. Portocarrero, F. C. Delicato, P. F. Pires, N. Gámez, L. Fuentes, D. Ludovino, and P. Ferreira. Autonomic wireless sensor networks: A systematic literature review. *Journal of Sensors*, 2014, 2014.

[13] A. Rowe, M. E. Berges, G. Bhatia, E. Goldman, R. Rajkumar, J. H. Garrett, J. M. Moura, and L. Soibelman. Sensor andrew: Large-scale campus-wide sensing and actuation. *IBM Journal of Research and Development*, 55(1.2):6–1, 2011.

[14] M. Salehie and L. Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14, 2009.

[15] M. Takruri, S. Challa, and R. Yunis. Data fusion techniques for auto calibration in wireless sensor networks. In *12th Int. Conf. on Information Fusion, 2009. FUSION'09.*, pages 132–139. IEEE, 2009.

[16] K. Tei and L. Gurgen. Clout: Cloud of things for empowering the citizen clout in smart cities. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 369–370. IEEE, 2014.

[17] P. Van Roy, S. Haridi, A. Reinefeld, J.-B. Stefani, R. Yap, and T. Coupaye. Self management for large-scale distributed systems: An overview of the selfman project. In *Formal Methods for Components and Objects*, pages 153–178. Springer, 2008.

[18] C. Vidal, C. Fernández-Sánchez, J. Díaz, and J. Pérez. A model-driven engineering process for autonomic sensor-actuator networks. *International Journal of Distributed Sensor Networks*, 2015, 2015.