DOI: 10.15439/2015F117
ACSIS, Vol. 5

A Quality Attributes Approach to Defining Reactive Systems Solution Applied to Cloud of Sensors

Artur Skowroński
Schibsted Tech Polska Sp. z o.o.
ul. Armii Krajowej 28
Krakow, Poland
Email: artur.skowronski@schibsted.pl

Jan Werewka
AGH University of Science and Technology
Department of Apllied Computer Science
Krakow, Poland
Email: werewka@agh.edu.pl

***Abstract*—Reactive systems have been investigated and used for a long time. Due to new methods and new technology development, the reactive systems needs their redefinition. These systems are currently an interesting topic for IT (Information Technology) solution providers. In this paper the authors try to define a new view of the architecture of reactive systems, because reactive systems are evolving and there is no clear definition of them. The starting point of the investigation was the reactive manifesto which defines reactive systems by four main features (quality attributes): responsiveness, resilience, elasticity and message driven interoperability. The mentioned quality attributes are the basis for developing a system solution. For each of the quality attributes, a set of tactics are proposed to maintain attribute required behavior. The suitability of the proposed tactics was investigated for Reactive Sensor Middleware which is part of a CoS (Cloud of Sensors) in the PaaS (Platform as a Service) layer. A cloud of sensors for pollution monitoring in urban areas was used as an example. Verification of the tactics has confirmed that some of the proposed tactics are suitable for the selected CoS subsystem.**

***Index Terms*—Keywords Reactive systems, reactive manifesto, software architecture, quality attributes, tactics, cloud of sensors, pollution sensing**

I. Introduction

IN THIS paper an approach is presented for the purpose of defining a software architecture model for a class of systems. The class considered here are reactive systems. In the classic book [1] on reactive systems design, systems are described by their characteristics: highly interactive, nonterminating process, interrupt driven, state-dependent response, environment-oriented response, parallel processes, usually stringent real-time requirements. The starting point of the analysis of reactive systems was the reactive manifesto [2], which defines reactive systems by its features. The manifesto stated that "a coherent approach to systems architecture is needed, and we believe that all necessary aspects are already recognized individually: we want systems that are responsive, resilient, elastic and message driven". The systems described by these features are called reactive systems. The reactive manifesto went through two revisions. The scalable and event driven traits from the previous version are replaced by elastic and message driven in the current version. We found this clarification interesting, especially when observing changes in dependency diagram which is part of both revisions. In the previous version the traits are all connected bidirectional lines, suggesting that all of them depends on each other, which was not very informative. That changed in Reactive Manifesto 2.0. The current iteration contains far more interesting inner dependencies. Message Driven pattern was defined as basis for all other traits. Responsiveness seems to be the main goal of reactive systems, because all other traits depends on it.

The goal of this paper is to develop an system solution based on selected quality attributes responsiveness, resilience, elasticity and message driven pattern. For the predefined set of quality attributes architectural tactics are proposed. The tactics will be used in determining architectural models. The approach is based on the classic software architecture development process proposed by Software Engineering Institute [3][4]. In the literature different solutions are proposed in the field. In [5] a general approach is proposed for embodying nonfunctional requirements (NFRs) into software architecture using architectural tactics. In [6] the influence of quality properties on decision making regarding software architecture was investigated.

II. Quality Attributes of Reactive Systems

Quality attributes are referred to as Non Functional Requirements (NFR) and represent a desirable behavior of the system and are key success factors in developing software architecture. In next subsections three quality attributes are considered: elasticity, resilience and responsiveness.

A. Elasticity

Elasticity is the ability of a system to scale resources up or down with minimal latency for different environment behavior during system runtime for different time periods. The ability can be reached manually or automatically. For the reactive systems the following sub attributes can be distinguished:

Consistent System Load. The load of IT systems can be not evenly distributed. The system should be able to scale both up and down - changing dynamically and automatically the amount of resources allocated. The goal is to handle user requests in a predictable, consistent manner. Attribute metrics should represent a standard level of usage of resources. It‘s lower boundary should not be set too low (the system is wasting resources, over-provisioned) while it‘s upper boundary should not be set to high - in this case, we are in constant risk of throttling the system at any moment (under-provisioning).

Latency in allocating and deallocating resources. The system should be able allocate resources to achieve elasticity corresponding to the current load. Due to that time is a critical criteria - the amount of time needed to set up an additional application process should be as short as possible. The application should be small and granulated and it‘s startup time really quick beginning from the "cold" system and ending with the ability of handling user requests. The criteria chosen are time of allocating and de allocating new resources. The attribute metrics are numeric - startup time from the state before allocating new resources to the moment, when new resources are able to handle new users.

Scalability. An elastic system should be scalable in a predictable way. Its performance should improve proportionally to added resources. The overhead of adding new application instance should be as small as possible. It is desirable to achieve linear proportion between those two values (performance to capacity of added resources).

B. Resilience

A resilient system is one that delivers a service that can be justifiably trusted when facing changes [9]. Resilience is related to a system's ability of maintaining service provision without deviating from the fulfillment of system goals, despite changes that might affect the system or its environment. An example of resilience evaluation is presented in [7]. For description of resilience the following sub attributes are selected based on the "technologies" defined in [9].

Evolvability. It is ability to respond effectively to change. Within evolvability, an important topic is adaptivity, i.e., the capability of evolving while executing and retaining the notion of justified confidence.

Assessability. It is based on verification and evaluation. Classically, verification and evaluation are performed offline in a pre-deployment stage. In reactive systems assessment and evaluation has to be performed at run-time, during operation.

Usability. Computing systems have already pervaded all activities of our life, hence the importance of usability.

Diversity. Diversity should be advantageous in order to prevent vulnerabilities, e.g. have single points of failure.

From our perspective, there are additional sub attributes which resilient system should have:

Automaticity. Reactive systems, due to constant changes, should not be administrated by people only. A system should react automatically on changes in it (e.g. a situation when a given service is down or a new deployment is ready) and perform a predefined strategy, mitigating the occurrence of human errors which could cause the system to close down. In a resilient system, infrastructure should automatically resize itself to keep required quality.

Rationality. The system should be able to provide value even when it is partially inaccessible. In that case the system is built from the isolated micro services, it shouldn‘t happen that the system is not responding when one trivial part is not able to respond (e.q. we have parts of system unaccessible due to power blackout). This feature cannot be easily added to the system in the later stages and should be taken into account during the design process. A system should know which parts are important and cannot be missed (responding to a failure) and which parts are trivial (it's just add value). Rational systems don‘t fail if there is no reason to do that.

C. Responsiveness

Responsiveness refers to the ability of a system to fulfill assigned tasks within a given time as seen by the user.

Consistent response time. It is important to achieve consistent response time for a system request. Typically, mean response time is used as a metrics. Unfortunately, it doesn‘t say much about a time consistency of a request. Two systems with exactly the same mean can have a highly different pattern of behavior. One system can be predictable and consistent, the other may have quick as well as slow response times. For both systems their mean metrics will still be similar. Consistency increase user trust in the system reliability. The attribute metrics may be a standard deviation of response time for a full request-response loop. It describes how a system behaves in the long term in a given time and presents information about the best and worst cases.

Adapting data processing to the given usecase. Data from the system should be accessible as fast as they bring the value for the end client. Topic becomes more important when we are talking about communication between systems which are better adapted to fast data acquisition and usage. Internet of Things presents both new opportunities and challenges, that's why Fast Data term becomes more and more important in synergy with Big Data systems.

III. Tactics for Reactive Systems

A tactic is a design decision that aims to improve one specific design concern of a quality attribute [5]. Software architects utilize a rich set of proven architectural tactics and patterns to help satisfy specific quality concerns. Architectural patterns have an overarching impact on a software system, and are typically selected early in the design process. They determine the overall style of the design and include well-known solutions schemes [8]. Tactics and patterns are known architectural concepts; the work [9] provides more specific and in-depth understanding of how they interact. In the next subsections tactics are proposed for a previously selected set of quality attributes.

A. Elasticity

1) Ability to scale part of a system independently: If we want to achieve usage of resources in a sufficient way, we need the ability to scale a different part of the system in response to its growing demand. Today, applications no longer rely on monolithic architecture. Parts of the system are developed in different technologies and they can have a very distributed necessity of system resources.

Sharding. Sharding is a specific type of database partitioning and its role is to split a database into smaller pieces, called shards, which are easier to manage, faster and less vulnerable to problems of global locking, meanwhile being easier to replicate due to its reduced size. Shards shouldn't share information between them, giving the ability to spread data between different physical instances and scale them independently, without the necessity of maintaining high-end, high-power systems. A common strategy is e.g. sharding data geographically, where we can take into account problem of latency too.

2) Decrease overhead of single components: Starting the executing of software components is the biggest factor in system latency. Below, are some tactics proposed, which are used to reduce software overheads

Containerization. Containerization and Software Containers are not new topics. However, they gain attraction in recent years, thanks to the support of IaaS Providers. In contrast to virtual machines it uses the kernel of the host machine and runs on isolated user space instances. Containers can be prepackaged and quickly distributed, with a minimal starting time of a single instance. It's worth to mention that there are particular Operating System Solutions created directly to work with Containerized software.

Lightweight technology stack. The bigger codebase, the longer load time can be felt by user. Splitting application into smaller pieces can drastically shorten time needed to run the new instance. Due to that fact, when want to achieve ability to run application instances dynamically on demand, we shouldn‘t relay on heavyweight enterprise technologies with huge amount of dependencies. Each application should have as little dependencies as possible and use external libraries/framework only when it‘s necessary.

3) Allowing for Resource Balancing: In the system which rely on the input from user, who can join or disconnect in any moment, amount of received data can vary dramatically over time. Having that in mind system should have ability for automatic resources balancing. Humans are error prone and too to slow to react while working with rapidly changing load. System should be able to allocate and deallocate resources to maximize ratio between throughtput and economic cost.

Implementing Backpressure. Backpressure in terms of responsive systems means the ability to acquire a feedback message from request passed through the system, e.g. by passing the message through the queue. Acquiring feedback is necessary to scale software in the correct way, e.g. the utilization of component resources. The implementation of backpressure is a nontrivial problem due to the asynchronous model of the communication. However, it is especialy important while dealing with the everlasting stream of data - there is possibility that the system under load will be overloaded by the incoming data.

Resource Managing based on Kernel Sharing Layer. An approach is used based on kernel features to provide the abstraction and isolation of system resources, instead of them on system-level defined rules, rather than monitoring the application and pooling its state through metrics values, such systems respond to application demands, by increasing its resources and spawning new instances (e.g. Apache Mesos [10] with Apache Aurora). It's solution fitted for Server Racks and Clusters to hide abstraction of multitenancy systems.

Immutable Designing. The less mutable state inside application, the easier application is to scale. In that case instances can‘t be easily replicated and a client can be served only by the instance with which started communicating. If the application is stateless, the load balancer can pass user input to whatever instance of application has free resources at the moment. The Domain Driven Design methodology is a good tool to evaluate which part of the application should be mutable and which not. Using good suited tools, such as functional programming languages or using message driven approach can be also benefitial.

B. Resilience

The resilience of the system is an offshoot of both dependability and availability, defined to better suit the demands of an architecture based on micro services. Lack of availability of microservices sums up. The important thing is to design a system in such way that when one of the elements fails it will not bring down the whole application functionality.

1) Replication: The most obvious way to achieve high availability of all systems parts is to provide redundancy. This is especially preferable when a considered system is stateless and every request can be processed by any instance.

Bulkhead. Bulkheads are used in ships to create separate watertight compartments which prevents the ship from sinking. The idea can be effectively used in computer systems. In a similar manner, a computer system should have redundant components which are easily replicable whenever something happens to the system and one of its counterparts.

2) Delegation: In contrast to the classic synchronous method calls, a reactive system cannot use exception and exception handling due to its isolated nature. Thrown exceptions don‘t have a chance to reach component which is able to handle it. Information about failure should be delegated to another component able to resolve it.

Feedback supervisor. In the reactive system a supervisor should exist, which is a special component existing outside the standard flow of the systems and has information about the whole system. Whenever a failure occurs, corresponding information with the whole bounded context should be send to the supervisor. It‘s a great way to decouple the standard flow of the system from the failure support and error handling mechanism (e.g. Netflix‘s Hystrix).

3) Isolation: The main problem of distributed systems is the possibility of partial failure. We don‘t want errors propagate over the systems, dragging whole infrastructure down. Isolation is an important trait of the system created from the micro services, which assures that failure of one part does not spread over whole system.

State and behavior containment by Bounded Context. Every component should be as small as possible and enclose specific problem domains inside the bounded context. Boundaries are connected by messaging protocols. This ensure that the system architecture reflects the problem domain making it easy to evolve. It also promote component composability and modularization on the architectural level.

Containment of failures. It should be possible to contain a failure inside the block where it occurred. The failure shouldn't be promoted to the next block, inhibiting "disease spreading". No error should be able to cascade through the system. Whenever we do not provide the fallback, we should fail-fast to not saturate system resources and pass failure to the supervisor.

C. Responsiveness

The responsiveness of the systems was investigated for a long time. In [5] six general principles for the synthesis of responsive software systems are presented: fixing, locality design, processing versus frequency tradeoff, shared resources, parallel processing, and centering. In the current systems developing a new approach for responsiveness tactics is essential.

1) Deferred data validation: The biggest difference between local software and software working throughout the network is the fact, that the distributed software has a far much longer feedback loop for each request. Whenever a user performs any action, it needs to wait to validate it on the server side, which can be a long operation striking user out of context. That's why it is important to sustain for the user an illusion of local work.

Normalization of data. System need to be able to cope with different type of inputs communicating on both a different protocols and data quality. That bring a necessity of bringing normalization layer which is able to retrieve common values from system, marking all data with an input specific metadata which can be used to additional analytics.

Data store synchronization. Thanks to better technology, it is possible to use data storage both on the client as on the server side. During work with web applications the user has feeling to work with native application by providing him with two data stores, a local and remote one. This is especially handy when both client and server are written in the same technology, sharing a common codebase. The user is working on the local copy bringing a short feedback. The local copy synchronization is performed in the background. The user is informed only whenever conflict happens.

Multiplayer game style data validation. To achieve smoothness of experience, programmers introduced a sophisticated system of the multiplayer game. Each player plays in his local environment and information about his actions is passed to the server, which confirms if its actions are possible to be done. The server has godlike power over each player and if he finds conflicts, resolves them and informs players about a verdict. Thanks to that each player has his own smooth experience.

2) Sustaining consistent response time: It is necessary to receive data as soon as they are able to be processed by system and end user. Synchronization should be done in the background. If our system responds to fast, responses it will be placed inside a buffer which should be maintained. If it will be too slow, we risk lowering the consistency of the overall system.

Streaming based data store To suit needs of the Fast Data system, our technology stack need to be adapted to processing not only huge amount of data, but also need to be able to process them in the fast way. Such a system needs to be able to combine storing huge amount of data from the many different concurrent inputs as a data warehouse, it also need to be able to deliver stream result in a quick way with a most current results for a waiting clients. An example of such a solutions can be Apache Storm and Apache Spark.

Non-blocking client-server communication. Communicating in a synchronous way is the biggest issue when trying to achieve consistent response times. A server communicating in a synchronous way is always waiting for a response and a one long request can delay a whole queue of operations. Each request should be responded to in the deferred way.

Worst case scenarios designing. To ensure that a response will be produced in reasonably, consistent time, it is important to use algorithms with low complexity. To provide a good level of stability in distributed systems, we need to bound a system with realistic timeouts. It is important not to break connections in case the data are processed correctly, but too long.

IV. A Cloud of Sensor with Reactive Sensor Middleware

The proposed tactics for the reactive system will be verified for RSM (Reactive Sensor Middleware) which is part of a CoS (Cloud of Sensors) in the PaaS (Platform as a Service) layer.

The proposed RSM is very important because it is assumed that the number of sensors will increase rapidly. The physical sensors and a well-defined communication interface will be delivered by sensor providers. The sensor providers will deal with service installation, sensor infrastructure maintenance, and the sensor data offering to sensor service consumers over the Internet. The service consumers can use the sensor data with their own or other providers' applications, which are integrated with the physical sensors. The described solution is known as a CoS (Cloud of Sensors). This makes it possible for users to lease the sensing hardware and associated applications instead of buying the complete infrastructure.

There are similar solutions to CoS such as cyber physical cloud (CPC), the Internet of things (IoT) or Fog Computing [11], which is a paradigm that extends Cloud computing to the edge of the network and services can be hosted at end devices. All these solutions [12] have sensors and a cloud as an integral part of their architecture. CPS consists of computer (cyber) systems that interact with the physical world. CPC is simply a CPS with cloud integration. Thus, CPC is CPS with a cloud as its backbone for computation and communication. In contrast, the Internet of Things (IoT) enables pervasive and ubiquitous interconnection in near real-time on a massive scale with different remote devices (mostly sensors) that can be uniquely identified, located, and communicated with. Cost effective and scalable IoT solutions can be achieved using cloud-centric architecture.

Cloud of Sensors Solution Architectures Existing research into CoS and related architectures proposes various solutions.

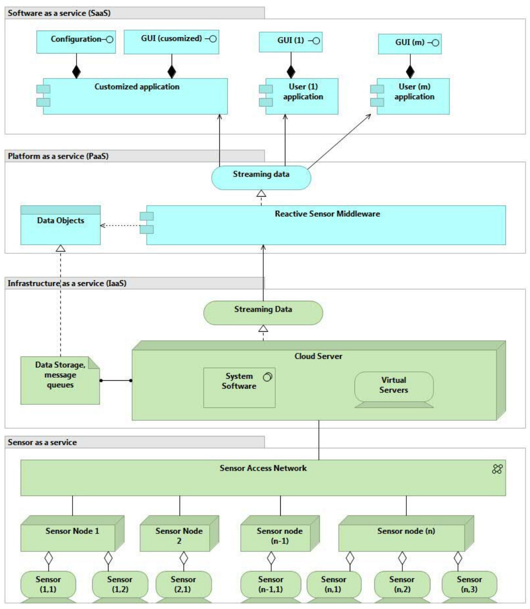

Fig. 1. CoS structure with Reactive Sensor Middleware located in PaaS layer

Some interesting examples are given below.

Paper [13] presents the design and evaluation of a Data Quality-Aware Sensor Cloud (DQS-Cloud) which is based on a cloud-based sensor data services infrastructure. The objective of the paper is to make a DQ as a multidimensional space in which all sensor devices produce multiple quality parameters or metadata such as accuracy, delay, frequency, latitude, longitude, sensor type, etc. Paper [14] proposes a new infrastructure called Sensor-Cloud infrastructure which virtualizes a physical sensor as a virtual sensor using cloud computing. An important issue is the developing of mathematical models (e. g. [15]) for the virtualization of sensor node resources. In [16] two alternative architectures for service management in IoT and sensor networks are discussed: based on Open Service Gateway (OSGi) framework and Remote Services for OSGi (R-OSGi) bundle.

For CPS systems other solution architectures are more suitable. In [17] a unified 5-level architecture is proposed as a guideline for implementation of CPS (Cyber Physical Systems). CPS solutions can be extended using SOA (Service-Oriented Architecture) solutions.

The CoS solution used for tactics verification of reactive systems, using cloud-centric architecture with Reactive Sensor Middleware in the PaaS layer is outlined on Fig. 1.

Pollution Monitoring in an Urban Area A cloud of sensors may have very broad applications. As an example an atmospheric pollution monitoring system was chosen in the paper. The goal of such a system is on-line pollution monitoring for decision making and the development of environmental policies, with the goal of reducing the impact of pollution on ecosystems and human health.

Some distinguished examples are given here. In [18] an interoperable system for air quality information management is proposed. The system is based on open-source standards-compliant tools and designed to develop a Spatial Data Infrastructure (SDI). In [19] a Pollution-Sense system for air pollution monitoring and control is presented. Participatory sensing combines the use of everyday mobile devices, such as cellular phones, GPS technology and location-based services, and sensors, to form interactive, bidirectional mobile sensing information systems. The system should provide large amounts of pollution data in time and space with different granularities. In [20] a method is described for the automatic detection of air pollution and fog using sensors mounted on vehicles. The described system consists of sensors which acquire their primary data from cameras and Light Detection and Recognition (LIDAR) instruments. In [21] a sensor cloud based on WSN (Wireless Sensor Networks) is proposed which virtualizes the wireless sensors and provides sensing as a service to users.

Different users of pollution monitoring systems can be distinguished. Some examples of users of such systems are: (1) Single persons or families (e.g. planning an excursion in an urban area); (2) People with allergies or breathing problems like asthma (e.g. planning to go outside); (3) Schools (e.g. planning different sport activities outside for children); (4) Early warning systems for municipal operation; (5) Urban planning used for air pollution reduction.

There are different types of pollution sensors which can be wearable, mounted on vehicles or UAVs, or installed in fixed positions (buildings, weather stations, etc.). The specialization of pollution sensors may differ significantly: (1) Personal environmental sensors, which are wearable sensors connected to or integrated with smart phones; (2) Pollution sensor stations installed on fixed points or on vehicles measuring air pollutants; (3) LIDAR (Light Detection and Recognition) used to detect small concentrations of air pollutants.

From the above discussion it is clear that providers of the sensor as a service layer can be individual persons or organizations with the motivation to receive payment for service proportional to sensor usage. The presented discussion aims only to be an overview of possible pollution system monitoring solutions, and is not intended to demonstrate the full picture.

V. VERIFICATION OF TACTICS

The proposed set of tactics will be verified for an RSM which is part of a CoS placed in the PaaS layer. Of course not all of the proposed tactics for reactive systems will be suitable for RSM. Only the tactics most useful while building the aforementioned system will be selected.

Elasticity is a very important aspect of a system based on dynamically attached external data sources (in our case sensors). We do not know in advance the size of the streams of data that our application will be exposed to. For this reason correctly implemented resource balancing needs to be a core part of the infrastructure. Proper implementation of backpressure tactics is especially important as the system needs to be able to control the flow of data. A perfect solution would appear to involve quickly running and disabling new application instances when there is an urgent load increase or decrease, in order to setup and disable containers on the shared Resource, especially when the application is stateless and implemented using a light technology (a good example of such a stack is Node.js Amazon Lambda). In this case, for a cloud of sensors a sharding tactic should be proposed. Usually data received from sensors are closely related to geographical locations or data sensor types, therefore the data should be logically grouped to ensure processing efficiency. This makes it possible to easily separate data in order to mitigate latencies during querying of the data store.

Resilience is defined as the time taken by a system to return to an acceptable state after failure. The investigated cloud of sensors is highly dependent on the external data, therefore constant supervision is essential. A component which monitors the behavior of intermediate elements is needed, and this can be achieved by using a feedback supervisor tactic. A good level of isolation in the system should be provided as data can be acquired from different inputs and the instances collecting the data are independent. A containment of failure tactic can be used to prevent the failure of one instance impacting others. All instances should not be deployed in a single cluster, instead the system should use a bulkhead tactic which splits it into different physical locations, thereby mitigating the potential for a critical situation in which the system is unresponsive as a whole.

An important aspect of the design of the discussed sensor cloud from a **responsiveness** perspective is the use of a normalization of data tactic. This ensures that the data store always has data in a proper, common format and is able to execute all necessary analytic operations quickly and responsively, without intermediate steps. Also, providing non-blocking client-server communication is important if we want to achieve consistent response times for clients of our cloud solution.

VI. Summary

The current reactive system's needs must be examined from different perspectives due to the emergence of new system solutions and new technologies. One such factor is the big data processing issue, caused by the need for real time data stream acquisition and visualization, which makes it important to perform architecture refactoring of reactive systems. The approach presented, which starts with the analysis of quality attributes and their tactics, seems proper at this stage of development. The investigation of the suitability of the proposed tactics for Reactive Sensor Middleware which is a part of CoS (Cloud of Sensors) placed in the PaaS (Platform as a Service) confirms the approach. The next stage will be experimentation on other real system examples, analyzing suitable architecture patterns and finally defining a reference architecture model for reactive systems.

References

[1] R. J. Wieringa, "Design methods for reactive systems," 2003.

[2] "The reactive manifesto, published on september 16 2014. (v2.0), http://reactivemanifesto.org." [Online]. Available: http://reactivemanifesto.org

[3] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Boston, MA, USA: Addison-Wesley, Inc., 1998. ISBN 0-201-19930-0

[4] R. Wojcik and et al., "Attribute-driven design version 2.0, tr-023." SEI, Carnegie Mellon Univ, 2014.

[5] S. Kim, D.-K. Kim, L. Lu, and S. Park, "A tactic-based approach to embodying non-functional requirements into software architectures." in *EDOC*. IEEE Computer Society, 2008. ISBN 978-0-7695-3373-5 pp. 139–148. [Online]. Available: http://dblp.uni-trier.de/db/conf/edoc/edoc2008.html#KimKLP08

[6] H. R. E. Majidi, M. Alemi, "Software architecture: A survey and classification," *2010 Second International Conf. on Communication Software and Networks*, pp. 460–464, 2010.

[7] J. Cámara, P. Correia, R. de Lemos, and M. Vieira, "Empirical resilience evaluation of an architecture-based self-adaptive software system," ser. QoSA '14. New York, NY, USA: ACM, 2014. doi: 10.1145/2602576.2602577. ISBN 978-1-4503-2576-9 pp. 63–72. [Online]. Available: http://doi.acm.org/10.1145/2602576.2602577

[8] J.-C. Laprie, "From dependability to resilience," in *38th IEEE/IFIP Int. Conf. On Dependable Systems and Networks*, 2008.

[9] N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? a model and annotation," *J. Syst. Softw.*, vol. 83, no. 10, pp. 1735–1758, Oct. 2010. doi: 10.1016/j.jss.2010.04.067. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2010.04.067

[10] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, "Mesos: A platform for fine-grained resource sharing in the data center," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2010-87, May 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-87.html

[11] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proceedings of the 2015 Workshop on Mobile Big Data*, ser. Mobidata '15. New York, NY, USA: ACM, 2015. doi: 10.1145/2757384.2757397. ISBN 978-1-4503-3524-9 pp. 37–42. [Online]. Available: http://doi.acm.org/10.1145/2757384.2757397

[12] V. Sehgal, A. Patrick, and L. Rajpoot, "A comparative study of cyber physical cloud, cloud of sensors and internet of things: Their ideology, similarities and differences," in *Advance Computing Conference (IACC), 2014 IEEE International*, Feb 2014. doi: 10.1109/IAdCC.2014.6779411 pp. 708–716.

[13] A. Kothari, V. Boddula, L. Ramaswamy, and N. Abolhassani, "Dqs-cloud: A data quality-aware autonomic cloud for sensor services," in *Collaborative Computing: Networking, Applications and Worksharing, 2014 International Conference on*, Oct 2014, pp. 295–303.

[14] M. Yuriyama and T. Kushida, "Sensor-cloud infrastructure - physical sensor management with virtualized sensors on cloud computing," in *Network-Based Information Systems (NBiS), 2010 13th International Conference on*, Sept 2010. doi: 10.1109/NBiS.2010.32. ISSN 2157-0418 pp. 1–8.

[15] S. Misra, S. Chatterjee, and M. Obaidat, "On theoretical modeling of sensor cloud: A paradigm shift from wireless sensor network," *Systems Journal, IEEE*, vol. PP, no. 99, pp. 1–10, 2014. doi: 10.1109/JSYST.2014.2362617

[16] D. Wilusz and J. Rykowski, "Comparison of architectures for service management in iot and sensor networks by means of osgi and rest services," in *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, ser. Annals of Computer Science and Information Systems, M. P. M. Ganzha, L. Maciaszek, Ed., vol. 2. IEEE, 2014. doi: 10.15439/2014F324 pp. pages 1207–1214. [Online]. Available: http://dx.doi.org/10.15439/2014F324

[17] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, no. 0, pp. 18 – 23, 2015. doi: http://dx.doi.org/10.1016/j.mfglet.2014.12.001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S221384631400025X

[18] F. D'Amore, S. Cinnirella, and N. Pirrone, "Ict methodologies and spatial data infrastructure for air quality information management," *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, vol. 5, no. 6, pp. 1761–1771, Dec 2012. doi: 10.1109/JSTARS.2012.2191393

[19] D. Mendez, A. Perez, M. Labrador, and J. Marron, "P-sense: A participatory sensing system for air pollution monitoring and control," in *Pervasive Computing and Communications Workshops, 2011 IEEE International Conference on*. doi: 10.1109/PERCOMW.2011.5766902 pp. 344–347.

[20] P. Sallis, C. Dannheim, C. Icking, and M. Maeder, "Air pollution and fog detection through vehicular sensors," in *Modelling Symposium (AMS), 2014 8th Asia*, Sept 2014. doi: 10.1109/AMS.2014.43 pp. 181–186.

[21] S. Madria, V. Kumar, and R. Dalvi, "Sensor cloud: A cloud of virtual sensors," *Software, IEEE*, vol. 31, no. 2, pp. 70–77, Mar 2014. doi: 10.1109/MS.2013.141