Cataloging design patterns for Internet of Things artifact integration

Rafał Tkaczyk*, Katarzyna Wasielewska*, Maria Ganzha†*, Marcin Paprzycki‡*,
Wiesław Pawłowski§,*, Paweł Szmeja*, Giancarlo Fortino¶,
*Systems Research Institute Polish Academy of Sciences, Warsaw, Poland
Email: firstname.lastname@ibspan.waw.pl
†Warsaw University of Technology, Warsaw, Poland
‡Warsaw Management Academy, Warsaw, Poland
§University of Gdańsk, Gdańsk, Poland
¶University of Calabria, Rende, Italy
Email: g.fortino@unical.it

***Abstract*—While Internet of Things (IoT) systems/applications/platforms/devices materialize with increasing speed, software engineering “reflection” does not follow “fast enough”. The situation is particularly “unbalanced” when one considers integration of independently developed IoT artifacts. To address this problem, we attempt at cataloging *software design patterns* that materialize in the context of interoperability of/within IoT ecosystems. The aim of this contribution is to briefly describe most common patterns (based on results of the INTER-IoT project), including analysis of common issues, and elaboration of a need for the creation of new (or extending existing) patterns in order to achive solutions applicable for IoT artifact integration.**

I. Introduction

Recently, one can observe re-emergence of sizable gap between “foundations of software engineering” and development of distributed systems. Specifically, this is visible in the design and implementation of Internet of Things (IoT) ecosystems (consisting of, among others, applications/systems/platforms/devices). While it can be claimed that the development of scalable, and inherently heterogeneous, IoT environments does not need software engineering, we believe that such claims are shortsighted and do not take into account importance of meta-level reflection, brought by software engineering. Therefore, within the scope of the INTER-IoT project [1] we have decided to delve into software engineering aspects of creation of interoperable IoT ecosystems. Out of multiple facets of software engineering, we start from identifying *design patterns* that materialize when forming IoT ecosystems, consisting of heterogeneous artifacts. The context of this work, and a source of reflection, is provided by use cases of the INTER-IoT project. Note that, while patterns were identified within the scope of the project, they generalize easily.

Before proceeding, let us note that the *design pattern* is understood as a general reusable solution to a problem that recurs repeatedly within a specific context in software design. It is a written document (based on a template) describing how to solve a problem that can be used in multiple situations. In this way, design patterns are formalized best practices. Their purpose is to increase re-usability and quality of code, while reducing the effort of development of software systems ([2], [3], [4]). Furthermore, a *pattern catalog* is a collection of related patterns, subdivided into a (small) number of categories. Here, note that some amount of cross referencing between patterns is natural in a pattern catalog [2].

Initial version of the proposed pattern catalog, and the process of it’s design is presented in what follows. Section II describes the methodology applied to pattern identification and description. Next, Section III presents results of examination of common, well-known pattern catalogs. Section IV outlines the format, used to describe identified patterns. Section V presents the obtained catalog and finally, Section VI concludes the text.

II. IoT interoperability design pattern identification – methodology

The proposed pattern catalog is based on the analysis of already existing solutions, good practices, and knowledge and experience of specialists working within the area. Let us now outline steps of the pattern identification process (detailed description can be found in[1]).

Step 1. Analysis of the state-of-the-art (SotA) (see, Section III) by examining common, well-known, pattern catalogs and extracting knowledge (e.g. verifying applicability in the IoT domain) useful to identify issues unsolvable by already defined solutions. Note that, SotA analysis included not only IoT pattern catalogs but also catalogs dedicated to, e.g., integration and SOA.

Note that while the design of IoT artifacts can apply existing design patterns, there are no *formal* guidelines for the IoT integration. Therefore, following the INTER-IoT approach we have decided to conceptually decompose IoT ecosystems into the following layers: Device-to-Device (D2D), Networking-to-Networking (N2N), Middleware-to-Middleware (MW2MW), Application & Services-to-Application & Services (AS2AS), Data & Semantics-to-Data & Semantics (DS2DS), and CROSS-Layer and use them in our analysis ([5]).

[1]D5.1 INTER-IoT Design patterns for Interoperable IoT Systems, INTER-IoT public deliverable, available June 2018.

Step 2. Analysis of IoT artifacts' integration process. Here, results were compared with those from *Step 1*, in order to extract the initial set of *new* patterns, related to integration on different layers. The use cases considered in the INTER-IoT project have also been analyzed, searching for procedures, which could be generalized and formalized to become patterns. It was decided that the initial catalog will follow the proposed structure, providing patterns for the identified layers. Note that most new patterns were inspired by multiple sources (including paradigms/technologies, e.g. *flow–based programming*).

Step 3. Detailed analysis of the initial set of patterns, with focus put on patterns related to integration/interoperability of/between IoT artifacts. As a result, the set of patterns was narrowed down (by eliminating patterns not related to integration).

Step 4. Final version of the catalog, using the *design pattern format*, defined to describe the identified patterns.

III. Analysis of the State of the Art

The main intention of SotA analysis was to identify patterns "useful for" IoT *integration*, understood as the ability of two (or more) IoT artifacts to exchange data, and trigger appropriate actions. Selected/created patterns should support development of integrated IoT ecosystems. Therefore, the most crucial were design patterns for: (i) integration/communication, (ii) security, (iii) architecture of software components, and (iv) domain use case solutions. This section summarizes inspected pattern catalogs. The short description includes: (i) catalog names, (ii) references, and (iii) short analysis (details can be found in the aforementioned deliverable).

1) *Object-oriented Patterns*, proposed by "Gang of Four" (GoF; [3], [4]), are fundamental in software engineering (object-oriented design theory and practice). In our context, GoF is a reference for software components developed during IoT artifacts integration. Moreover, many pattern description formats are based on the one proposed by GoF, because its structure is clear and intuitive.
2) *Enterprise Integration Patterns (EIP)*, are an extension of GoF patterns, and provide guidance when integrating or designing distributed systems [7]. In particular, patterns such as "Message transformation", "Messaging Pattern", "Messaging Routing" are of interest.
3) *Service-Oriented Architecture (SOA) Patterns*, including *Micro-services*, are related to: (i) solutions within AS2AS and MW2MW layers, (ii) characteristics of services exposed by IoT artifacts that join the ecosystem, (iii) service orchestration [8], [9].
4) *Reactive Patterns* are usable in any distributed application, and thus also for IoT artifacts integration. "Messages", "Message Flow" and "Flow Control" patterns are of particular interest here [10], [11].
5) *Agent-oriented Design Patterns*, could be used to allow integration, interconnection, and interaction between agent-based and/or non-agent software components and systems [12], [13], [14]. In particular, the "Facilitator" and "Agent Proxy" patterns by Kendall [14] can be useful/effective to support the design of gateways and proxies between subsystems (or layers) of IoT systems.
6) *IoT Patterns*, identified in [15], [16] are common to all IoT solutions. However, *none* of found catalogs is related to the integration of *existing* IoT artifacts (they are mainly focused on the deployment of *new* IoT systems).
7) *Security Patterns* deal mostly with authentication and authorization, as well as secure communication [17], [18]. However, they do not deal with security related to interoperability within IoT ecosystems.

IV. IoT integration pattern format

Different design patterns specifications have been proposed. They usually include a narrative text, with a predefined structure. Based on [19], as well as work of GoF ([3], [4]), the following pattern description has been used in our work:

- *Pattern name* – unique name of the pattern.
- *Inspired by* – name(s) of pattern(s) that a given one is based on / extends. In most cases, when pre-existing patterns did not fully solve specific problems, new patterns were created, extending existing ones.
- *Related patterns* – other patterns, related to the given one.
- *Intent (summary)* – short description of the goal behind the pattern and the reason for using it (an extension of the "Pattern name", explaining its action/purpose).
- *Problem & Solution* – scenario that illustrates a problem and how the pattern solves it.
- *Applicability* – situations, in which the pattern is usable; context for the pattern.
- *UML representation* – structure of the pattern modeled with a UML diagram (mostly deployment and component diagrams).
- *Implementation* – extension of the "UML representation" property, i.e. description of realization and architecture (not a source code, like in the GoF).
- *Known uses* – an example usage of the pattern within the INTER-IoT pilot installation.

Due to space limitations, fields: "UML representation", "Implementation", and "Known uses" have not been included in what follows (find them in INTER-IoT deliverable D5.1).

V. Design patterns for IoT artifacts integration – final catalog

This section summarizes the identified patterns related to IoT artifacts integration.

A. D2D Layer

Pattern name: *IoT Gateway Event Subscription*
Inspired by: (1) "Publish/Subscribe" (IoT Patterns: Design Patterns for Interaction), (2) "Publish-Subscribe Channel" (EIP: Messaging Channels), (3) "Facilitator", and (4) "Proxy" (Agent Design Patterns: by Kendall).
Related patterns: (1) "D2D REST Request/Response".
Intent: D2D gateway allows data forwarding (any type).

Flexibility in the D2D layer is achieved by decoupling a gateway into: a physical part that handles network access and communication protocols, and a virtual part dealing with remaining gateway operations and services. Note that, in this way, data providers (communicating within the network) and their identities (known to the virtual layer) can also be decoupled.
Problem & Solution: To provide interoperability between two heterogeneous IoT devices, the solution should establish bidirectional, asynchronous communication with the ability to publish, filter and consume data. Here, the IoT gateway is used as a subscription mechanism. It is an intermediary between IoT artifacts (in D2D communication, between two devices). It allows transmission of data generated by "sensors" to the destination, and asynchronous messaging between artifacts that interact with it. If required, the gateway should perform protocol conversion to enable communication. Senders of messages (publishers) do not program messages sent directly to specific receivers (subscribers). Instead, they publish them, using defined classes, without knowledge of subscribers. Similarly, subscribers express interest in one or more classes and receive only messages of interest (without knowing publishers). Significant is the structure of the message, which should contain subscription information (e.g. message endpoint; see: "D2D REST Request/Response" pattern).
Applicability: Used within event-based communication, when asynchronous data is to be pushed/pulled to/from the gateway.

Pattern name: *D2D REST Request/Response*
Inspired by: (1) "Request-Response" (Reactive Patterns: Message Flow), (2) "Request-Reply" (EIP: Messaging Patterns), (3) "Request/Response" (IoT Patterns: Design Patterns for Interaction).
Related patterns: (1) "IoT Gateway Event Subscription".
Intent: A request/response solution for gateway communication within the D2D layer.
Problem & Solution: IoT Gateway needs to communicate with IoT artifacts. It should be accessible to authorized external elements to enable reception of information and execution of control and configuration orders. For example, the main goal of IoT ecosystems is to allow heterogeneous IoT artifacts to retrieve information. Thus, the artifacts's middleware should be able to communicate with the IoT gateway to enable needed information flows. Thus, it is desirable to connect IoT artifacts (if possible) through a HTTP/REST API using the Request/Response pattern. This communication pattern allows message exchange, in which a requester (e.g. middleware or gateway) sends a request message to a replier system, which receives and processes the request (e.g. middleware or gateway), ultimately returning a message, in response.
Applicability: Used when communication between the middleware of an IoT artifact and the IoT gateway is performed through a REST API (middleware $\rightarrow$ gateway is typically based on Publish/Subscribe). Also, for management purposes, the gateway will expose a REST endpoint where configuration and management actions can be performed using the Request/Response patterns.

B. N2N Layer

Pattern name: *IoT Pattern for Orchestration of SDN Network Elements*
Inspired by: (1) "Software-defined networking (SDN) orchestration" ([23]), (2) "Network virtualization (NV)" ([24]).
Related patterns: none
Intent: Monitoring and configuration of SDN elements (virtual-switches) with an orchestrator component (Controller) exchanging flow and control messages. To provide interoperability between different domains connected to a network or between different network topologies and/or configurations.
Problem & Solution: Domain-focus of IoT deployments isolates them from each other. One of approaches to interconnection is, instead of realizing it at the device/gateway level, to move it to upper layers. In particular, at the network layer, interoperability and exchange of information can be achieved by applying pattern that manages elements of the network that provide connection from different domains to the network itself. The pattern is used in development of virtual SDNs, where all elements are virtual resources, or instances, controlled within a central point, or orchestrator. N2N interconnection can then be performed through the SDN. Different networks (in different locations), can be virtually interconnected and belong to a single Virtual LAN. Thus, physical separation of networks becomes "invisible", thus facilitating elastic definition of needed connectivity.
Applicability: Used when an IoT SDN is deployed, to enable its functionality. Allows total software control over network functions, and transparent N2N interoperability.

C. MW2MW Layer

Pattern name: *IoT Artifact's Middleware Simple Component*
Inspired by: (1) "Simple component" (Reactive Patterns: Fault Tolerance and Recovery).
Related patterns: none
Intent: Every IoT artifact is designed to perform (in full) a single task (single responsibility principle, where each class should have only one reason to change).
Problem & Solution: In complex systems with multiple functions, it may be necessary to have these functions performed by different components. Their responsibilities are to be divided recursively, until desired component granularity is reached. This enables testing, debugging and extending complex system more efficiently, simplifying all operations.
Applicability: Basic pattern that can be universally applied. Does not impose level of granularity to be achieved, but indicates that analysis should be performed in order to end up with the best component decomposition. Should be applied recursively, remembering to not to divide components too far, to avoid trivial ones.

Pattern name: *IoT artifact's Middleware Message Broker*

Inspired by: (1) “Message Broker” (EIP: Message Routing), (2) “Broker” (IoT Patterns: Design Patterns for Interaction).
Related patterns: (1) “IoT artifact’s Middleware Self-contained Message”.
Intent: Facilitates passing messages between IoT artifacts.
Problem & Solution: In middleware, composed of several independent components, point-to-point connections should be avoided, as they result in multiple interfaces that expose operations of each component. Furthermore, having point-to-point interfaces complicates dynamic reconfiguration, matching of security constraints, and quality of service (QoS) requirements management. Message broker helps to overcome those limitations by enforcing common messaging interface upon different components. This allows components to initiate interactions with other components, no matter their architecture and purposes. Each component communicates directly with the broker only, while within the broker, each component is represented with a logical name, making its internal operation hidden from other components. Crucial is the proper format of message, which consists of the payload and the label, storing information needed by the broker.
Applicability: Central Message Broker receives messages from multiple message producers, determines their destinations (message consumers), and routes them to channels specific for their destinations. Allows decoupling the sender from the destination and maintains central control over the flow of messages. This can be achieved through usage of topics, to which consuming components can: subscribe, and proceed to consume awaiting messages.

Pattern name: *IoT Artifact’s Middleware Self-contained Message*
Inspired by: (1) “Self-contained message” (Reactive Patterns: Message flow), (2) “Messaging Metadata” (SOA Patterns).
Related patterns: none
Intent: Messages contains complete information needed for execution of a specific action.
Problem & Solution: Within middleware, messages should be “pure and complete” representations of events (or commands), regardless when they are to be interpreted. Each middleware component must always be able to extract from the message, stored there, complete information needed for its routing and interpretation, with only minimal data stored within the middleware components. Each message has distinct set of types associated with it. Each middleware component processes and routes messages based solely on these types. For each message that travels “downstream”, there can be a response that travels “upstream”. Such messages might, for example, include additional response message type. Matching messages that go downstream with responses that go upstream can be done through remembering and distinguishing different chains of messages (conversations).
Applicability: Allows middleware components to be “context-free”, storing only a minimal information needed for message processing and routing. Can be also employed when there is no need to reference past messages, except for responses, and even then, these are only semantically linked to original messages (could exist without original messages).

Pattern name: *IoT Artifact’s Middleware Message Translator*
Inspired by: (1) “Message Translator” (EIP: Message Transformation), (2) “Data Format Transformation” (SOA Patterns).
Related patterns: none
Intent: Translation of messages to/from IoT artifact’s middleware internal message format and platform’s proprietary data models and data formats.
Problem & Solution: The purpose of the middleware is to pass information between different IoT artifacts. However, artifacts produce/consume messages in “their” formats and data models. Hence, to make sense of exchanged messages, they have to be syntactically and semantically translated. A message translator enables conversion between proprietary data models and data formats, used by artifacts, and the internal data model and data format, used by IoT middleware components.
Applicability: Enables interoperability between different platforms without the need to introduce translations between every possible pair of platforms, i.e. translation into and out of the common INTER-IoT data model [25]. Semantic translation from and into the internal message format is done by a dedicated IoT semantic translation component, while syntactic translation is completed in bridges to/from artifacts, as only they know the internal data syntax.

D. AS2AS Layer

Pattern name: *AS2AS Flow-based Service Composition*
Inspired by: (1) “Flow–based programming” ([26]).
Related patterns: (1) “AS2AS Service Orchestration”, (2) “AS2AS Discovery of IoT Services”
Intent: Generate execution flow that allows interoperation and composition of services from different IoT artifacts.
Problem & Solution: Pattern necessary to define execution flow that allows specific sequence of execution of multiple IoT services. Flow–based programming (FBP) defines applications and services as networks of “black box” processes, which exchange data across predefined connections by message delivery, where connections are specified externally to processes. Considered pattern allows creation of sequential execution flows using those services, thus allowing composition among different IoT services. Black boxes that represent IoT services can be linked by wiring the output of a service with the input of a different one (output messages from a service are routed to another service input). Thus, by wiring IoT services execution flow can be instantiated.
Applicability: Used in black box representation of IoT services to be interconnected through an FBP link, generating a flow.

Pattern name: *AS2AS Service Orchestration*
Inspired by: “Service Orchestration” (SOA Patterns).
Related patterns: none

Intent: To specify orchestration of services to facilitate interactions among different IoT services.
Problem & Solution: Cooperating, diverse, heterogeneous IoT artifacts involve huge number of different services that have to work together. Important is not only the message flow from point(s) to point(s) but also triggering necessary actions (during the flow). The common problem is that existing processes/actions are duplicated (not reused). This pattern allows union and orchestration of heterogeneous IoT services, creating a specific process. The main idea is to define a set of IoT nodes, i.e. services and interfaces that run within the integrated platforms. Internal, central, element wires nodes necessary to handle the specific task and controls processes.
Applicability: Reuse of process fragments. Orchestration enables composition of IoT service workflows, based on services from IoT artifacts.

Pattern name: *AS2AS Discovery of IoT Services*
Inspired by: (1) "Discovery" (IoT Patterns: Design Patterns for Interaction), (2) "Enterprise inventory" (SOA Patterns).
Related patterns: none
Intent: Registering and claiming specific services, used by the artifacts (within the IoT ecosystem).
Problem & Solution: Multiple IoT services, from different IoT platforms, provide a wide range of functionalities that have to be discoverable, to be aware of them and to use them. This pattern enables registration of services (in a service catalog), in order to find them and (potentially) use through an AS2AS layer solution. Here, only registered services, indicating their associated features, can be discovered.
Applicability: Pattern for providing service interoperability via registration and service retrieval. Applied to services that run within the IoT ecosystem, and used by other IoT artifacts.

E. DS2DS Layer

Pattern name: *Alignment-based Translation Pattern*
Inspired by: (1) "Message Translator" (EIP: Message Transformation), (2) "Data Model Transformation" and (3) "Metadata centralization" (SOA Patterns), (4) "Market Maker" (Agent Design Patterns).
Related patterns: none
Intent: Semantic translation of RDF messages exchanged between IoT artifacts, based on alignments (sets of correspondences) defined between artifacts' ontologies.
Problem & Solution: Building the IoT ecosystem involves combining existing solutions, which (likely) belong to different owners and have been developed using different technologies (e.g. Web Services "combined with" a graph database, communicating using JSON messages, to exchange information with application that uses XML messages). Consequently, they differ both on syntactic and semantic level. Interoperation between artifacts should be achieved regardless of the underlying technology. Without loss of generality, we assume RDF message format, since other formats can be transformed to RDF. Semantics of messages is artifact specific (ontology can be natively supported, or semantics, e.g. expressed in XSD, can be lifted to an OWL ontology). Hence, for semantic interoperability, a method for defining correspondences should support mapping between specific URIs, parts of the RDF structure, transformations etc. The component implementing the translation should provide interfaces to submit messages to be translated and publish translated messages.
Applicability: Providing semantic translation between RDF messages exchanged between heterogeneous IoT artifacts. Translation, based on one-to-one translation (alignment), should be possible to define for any two artifacts.

Pattern name: *Translation with Central Ontology*
Inspired by: (1) "Message Translator" (EIP: Message Transformation), (2) "Data Model Transformation" and (3) "Metadata centralization" (SOA Patterns), (4) "Market Maker" (Agent Design Patterns).
Related patterns: none
Intent: Semantic translation of RDF messages exchanged between IoT artifacts, where one involves central/common data model.
Problem & Solution: To provide common understanding in the semantic translation process a modularized central ontology can be created on the basis of IoT and domain ontologies. Here, a domain ontology is a conceptual model for a specific domain, e.g. transportation, health, etc. IoT ontology describes different IoT aspects, e.g. platforms, devices, sensors, services, etc. Central ontology should reuse / be based on existing standards (e.g. SSN, SOSA, SAREF, etc.). This approach is highly scalable: it is possible to add artifacts to the existing IoT ecosystem by instantiating translations with the central ontology (i.e. creation of alignments; see above). This approach requires less preparation/work, as only a single "point of joining" has to be instantiated. Furthermore, the long-term maintenance is simplified, as changes in a single platform require localized adjustments only. The component implementing the translation should provide interfaces to submit messages to be translated and publish messages after translation (realizable via an appropriate pattern, above).
Applicability: Providing semantic translation between multiple heterogeneous IoT artifacts that are to exchange RDF messages.

F. CROSS Layer

Pattern name: *IoT SSL CROSS-Layer Secure Access*
Inspired by: (1) Security Patterns, (2) IoT Patterns: Design Patterns for IoT Security.
Related patterns: (1) "Login Authentication", (2) "Sensitive Data Encapsulation", (3) "Encryption and Single Point of Access".
Intent: Ensuring security of interactions with external interfaces (i.e. APIs) of every layer of the IoT ecosystem.
Problem & Solution: As IoT architecture is composed by diverse layers, access to each of them, as well as interactions between them, must be secure. To ensure a sufficient level of

security on each of the IoT layers, different security mechanisms can be implemented: authentication of credentials, use of authentication tokens, or Secure Sockets Layer (SSL). In an IoT ecosystem, layer access will be secured with the SSL that employs the IoT SSL pattern. Every IoT layer exposes a REST API that represents an external interface accessible to the outside actors, such as other IoT layers, users, or IoT artifacts. To enable use of such APIs to only allowed actors the access is secured through SSL. REST APIs are accessible through a browser, which should provide a trusted certificate. Only after the establishment of a secure connection authentication through login will be allowed, to open the access to the layer API. Further operations on the layer API will be done using this secure connection.
Applicability: Pattern applied in interactions of any actor with the IoT environment layer's APIs. Access can also be done internally between pairs of different layers.

It should now be clear that, for the development of IoT integration patterns, all catalogs, described in Section III, provided inspiration in defining the final set of design patterns. Here, the most often used catalogs were: *EIP* and *SOA Patterns*, because they described the most common issues, related with communication, cooperation, etc. Moreover, while the *Object-oriented Pattern* was used only as an inspiration, the format of the GoF catalog was crucial in defining presentation format used here. Finally, note that catalogs (described in SotA) were not the only source of ideas in defining new design patterns. Other sources were also common paradigms and technologies, i.e.: "Software – defined networking orchestration", "Network virtualization", "Flow–based programming" and "Web API Design: Crafting Interfaces that Developers Love".

VI. Concluding remarks

This note summarized the process of defining IoT artifacts integration patterns, starting from the SotA analysis to the summary of the content of the current catalog. Every pattern was described using format, which was created on the basis of the GoF templates. Pattern descriptions outline the proposed solution, as well as the reasons for creating a new pattern, instead of using existing approaches. Moreover, inspirations and example of usage have been presented. Obviously the above catalog can be further extended as a result of future work in the domain.

Acknowledgment

This research was partially supported by the European Union's "Horizon 2020" research and innovation programme as part of the "Interoperability of Heterogeneous IoT Platforms" (INTER-IoT) project under Grant Agreement No. 687283.

References

[1] INTER-IoT Project homepage. [Online] Available: http://www.inter-iot-project.eu (2017)
[2] F. Buschmann, "Pattern–oriented Software Architecture: A System of Patterns", 1996 John Wiley & Sons, Inc., New York, NY, USA.
[3] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design Patterns: Elements of Reusable Object-oriented Software", 1955 Boston, MA, USA: Addison–Wesley Longman Publishing Co., Inc.
[4] E. Gamma, R, Helm, R. Johnson, L. O'Brien, "Design Patterns 15 Years Later: An Interview with Erich Gamma, Richard Helm, and Ralph Johnson", 2009 Software Development & Management.
[5] G. Fortino, C. Savaglio, C. E. Palau, J. Suarez de Puga, M. Ganzha, M. Paprzycki, M. Montesinos, A. Liotta, and M. Llop, "Towards Multi-layer Interoperability of Heterogeneous IoT Platforms: The INTER-IoT Approach", 2017 Integration, Interconnection, and Interoperability of IoT Systems. Internet of Things (Technology, Communications and Computing). Springer, Cham
[6] G. Hohpe, B. Woolf, "Enterprise Integration Patterns", 2004 Boston, MA, USA : Pearson Education Inc.
[7] (2017) Enterprise Integration Patterns. [Online] Available: http://www.enterpriseintegrationpatterns.com/patterns/messaging/toc.html
[8] (2017) Service Oriented Architecture Patterns. [Online] Available: http://soapatterns.org/introduction
[9] (2017) A pattern language for microservices. [Online] Available: http://microservices.io/patterns
[10] (2017) The Reactive Manifesto. [Online] Available: http://www.reactivemanifesto.org
[11] R. Kuhn, B. Hanafee, and J. Allen, "Reactive Design Patterns", 2016, Manning Publications.
[12] Y. Aridor, and D. Lange, "Agent Design Patterns: Elements of Agent Application Design", 1998, AGENTS '98 Proceedings of the Second International Conference on Autonomous Agents, New York, NY, USA, pp. 108–115.
[13] D. Duego, M. Weiss, and E. Kendall, "Reusable Patterns for Agent Coordination", 2001, [red.] F. Zambonelli, M. Klusch and R. Tolksdorf A. Omicini. Coordination of Internet Agents: Models, Technologies, and Applications, Springer.
[14] E.A. Kendall, "Role Models: Patterns of Agent System Analysis and Design", 1999, ACM, Agent Systems and Applications/Mobile Agents (ASA/MA-99).
[15] M. Koster, "Design Patterns for an Internet Of Things A Design Pattern Framework for IoT Architecture", 2004 Available: http://community.arm.com/groups/internet-of-things/blog/2014/05/27/design-patterns-for-an-internet-of-things
[16] S. Qanbari, "IoT Design Patterns: Computational Constructs to Design, Build and Engineer Edge Applications", 2015, Berlin, Proc. of IEEE IoTDI.
[17] B. Escribano, "Privacy and security in the Internet of Thing: challange or opportunity", (2017) Available: http://www.olswang.com/media/48315339/privacy_and_security_in_the_iot.pdf
[18] J. Ajit, M.C. Sunil, "Security considerations for Internet of Things" (2017) Available: http://www.lnttechservices.com/sites/default/files/whitepapers/2017-07/whitepaper_security-considerations-for-internet-of-things.pdf
[19] M. Fowler, "Writing Software Patterns." (2006) Available: https://www.martinfowler.com/articles/writingPatterns.html
[20] (2017) Pattern Language Homepage Available: http://www.patternlanguage.com/
[21] T. Wellhausen, A. Fiesser, "How to write a pattern?: a rough guide for first-time pattern authors", 2011 EuroPLoP '11 Proceedings of the 16th European Conference on Pattern Languages of Programs, New York, NY, USA.
[22] N. B. Harrison, "Advanced Pattern Writing, Patterns for Experienced Pattern Authors", Available: http://europlop.net/sites/default/files/files/1_2003_Harrison_AdvancedPatternWriting.pdf
[23] (2017) Software-defined networking orchestration Available: https://www.sdxcentral.com/sdn/definitions/what
[24] (2017) Network virtualization Available: https://www.sdxcentral.com/sdn/network-virtualization/definitions/whats-network-virtualization/
[25] M. Ganzha, M. Paprzycki, W, Pawlowski, P. Szmeja, and K. Wasielewska, "Alignment-based semantic translation of geospatial data", Proceedings of 3rd International Conference on Advances in Computing, Communication & Automation (ICACCA 2017), IEEE Press, Los Alamitos, CA, (in press).
[26] (2017) Flow–based programming Available: http://www.jpaulmorrison.com/fbp/
[27] (2017) Web API Design: Crafting Interfaces that Developers Love Available: https://pages.apigee.com/rs/apigee/images/api