SDPS-2019
Printed in Taichung, Taiwan, July, 2019
©2019 Society for Design and Process Science

SOFTWARE ENGINEERING CHALLENGES IN REALIZING SMART-CITY SYSTEMS

Mehmet Aksit
FMT, University of Twente, Enschede and TOBB-ET University, Ankara
{m.aksit@utwente.nl}

ABSTRACT

More and more cities adopt the concept of smart cities to manage their processes and to satisfy increasingly sophisticated demands of their citizens. Unfortunately, due to the complexity and lack of effective support by current infrastructures, developing smart-city systems is a very tedious task. This paper identifies 6 categories of challenges as possible obstacles in the cost-effective realization of smart-city systems. For each obstacle, a possible research approach is suggested. In relation to the realization of smart-city systems, to the best of our knowledge, this is the first paper that presents these obstacles in a single publication.

INTRODUCTION

More and more cities adopt the concept of smart cities to manage their processes and to satisfy increasingly sophisticated demands of their citizens. Unfortunately, due to the complexity and lack of effective support by current infrastructures, developing smart-city systems is a very tedious task. Within this context, this paper discusses the software engineering challenges in realizing smart-city systems.

The concept of smart cities follows the evolutionary steps of automation (Ellis & Nutt, 1980). It aims at enhancing efficiency and effectiveness; It is first based on replacement of the current state by (new) ICT techniques and then causes a transformation to something different with respect to the past; It is a disruptive development. What is "smart" is a moving target and can only be defined relatively.

We classify smart-city systems as applications and infrastructures. Our focus is on infrastructures. Infrastructures support the development of applications and as such they play a crucial role in the development of applications. To increase the effectiveness and efficiency in realizing smart-city systems, as a part of the infrastructure, effective software engineering models, methods, techniques and tools must be provided so that software engineers (smart-city developers) are conveniently supported in their application development activities. The challenges that are discussed in this paper are due to the lack of effective support by the current infrastructures.

Infrastructure design refers to many different areas from sensor networks to business platforms. Much work has been carried out in this context and many useful sub-systems are already available for use. A smart-city infrastructure, therefore, must be designed as an enabler, integrator and harvester of the available technologies individually and together and must fill the gaps where it is needed.

We identify 6 categories of software engineering challenges in realizing smart-city systems. These are, (1) developing the necessary models for smart-city systems; (2) managing and optimizing clusters of systems; (3) designing models, methods and tools for critical infrastructures; (4) optimizing the necessary quality attributes by system adaptation at run-time; (5) integrating software; and (6) designing a smart infrastructure with a high degree of interoperability, configurability, adaptability and evolvability.

We discuss these challenges in some detail in the following sections. The purpose of this paper is to make the software engineers be conscious about the problems that they may face while developing smart-city systems and to inspire researchers in their research activities. For each identified challenge, we also present a research approach to address the challenge in systematical manner.

This paper is organized as follows. The next section gives definitions and emphasizes the importance of effective infrastructures in developing smart-city applications. The remaining sections present the challenges and propose research approaches to address these. Finally, the last section concludes the paper.

SMART CITIES

Recently, the concept of smart cities has attracted a lot of attention. A smart city is generally defined as "an urban area that uses different types of electronic data collection sensors to supply information which is used to manage assets and resources efficiently" (Wiki-a, 2019). The term smart cities, however, may refer to different definitions depending on the stakeholders involved (Gil-Garcia, Pardo & Nam, 2015). For example, viewpoints of stakeholders over smart cities who are in charge of managing public services, city administration, governance, ecological sustainability, ICT systems may differ from each other considerably. In addition, residents have also a strong interest in the services offered by a smart city.

There has been a considerable number of research and development activities in the world to create smart cities (Panos & Pardalos, 2017) (Bibri, 2018). Many cities around the globe claimed the status of a smart city (Sanseverino, Sanseverino & Vaccaro, 2018) (Smart Cities Council, 2019) (EU report, 2007). The topic of smart cities is also considered as a very strategic topic by many governments.

This paper particularly focuses on the generic infrastructure, which is broadly defined as a set of generic information technology (IT) components that are the foundation of IT services for smart cities. Here the term generic refers to reusable components that can be utilized by diverse applications. The infrastructure is divided into three layers:

1. Sensor network layer: It involves the design and deployment of sensors, communication protocols, data gathering and transmission, data interpretation, etc.
2. Architecture layer: It utilizes the services offered by the sensor network layer and implements the core software system architectures so that business platform layer applications can be developed.
3. Business platform layer: It utilizes the services offered by the other layers and provides end-user and/or business solutions to the smart-city requirements.

The functionalities of these layers are not fixed and evolve continuously with the advancements in technology and changing user demands.

Smart-city applications can be very diverse. Companies may offer different kinds of smart-city applications for a very diverse range of needs. Agility is an important feature for companies to develop smart-city applications (just) in time according to the changing end-user demands and market expectations. Infrastructures must support a large category of smart-city applications and be flexible enough to conveniently introduce new techniques when they are demanded.

The design of a generic infrastructure refers to a large category of publications in many different areas. These are for example, sensors and sensor networks (Ferrari, 2010), big data (Mohanty & et al., 2015) and machine learning (Goodfellow & et al., 2016), self-adaptive systems (Kounev & et al., 2017), service-oriented architectures (Dhara, Dharmala & Sharma, (2015), systems of systems (DoD, 2008), cloud computing (Marinescu, 2017), cyber-physical systems (Suh & et al., 2014), cyber-security (Lehto & Neittaanmäki, 2015), critical infrastructures (Gritzalis, Theocharodou & Stergiopoulus, 2019), reliable systems (Birolini, 2017), dependable systems (de Lemos, Gacek & Romanovsky, 2003), business platforms such as decision support systems, business process modeling (Shishkov, 2018), and a large set of software engineering literature on programing languages, software architectures, model-driven architectures, testing, verification etc. (Sommerville, 2016) (van der Linden & et al., 2007)(Da Silva, 2015)(Nielsen, 2014). Many useful systems with different capabilities are already available for use. The infrastructure therefore, must be designed as an enabler, integrator and harvester of the available technologies individually and together and must fill the gaps where it is needed.

Economical, sustainable and robust software systems which fulfil functional and qualitative requirements are essential for all software systems (Akşit, 2018). Achieving these generic objectives are considered as the main objective of the Software Engineering discipline (Sommerville, 2016). To fulfill the requirements of smart-city applications now and in the future, software engineering models, methods, techniques and tools are crucial. No matter how intelligent a software solution is, if it cannot be realized with the desired quality attributes, one cannot expect an economical value out of it (Akşit, 2018). The software engineering challenges presented in this paper, therefore, form an important obstacle in cost-effective realization of smart-city applications.

In addition to offering diverse platform services, infrastructures must also provide effective software engineering methods and techniques so that software engineers (smart-city developers) are conveniently supported in their application development activities.

LIMITATIONS OF THE STATE-OF-THE-ART TECHNIQUES AND APPROACHES TO ADDRESS THESE

Unfortunately, due to the complexity of the systems, the necessity of coordinating, integrating and optimizing different systems together, dealing with different levels of abstractions from sensor networks to business platform layers, etc., developing smart-city applications is generally a very tedious and time consuming task. Currently a considerable number of research activities is being carried out within this context (Bibri, 2018). Along this line, after carrying out an extensive study on the related literature on infrastructures and interviewing the relevant persons, we have identified 6 general categories of challenges:

Challenge 1: Developing models for smart cities

Many software companies specialize in certain application areas. In such cases, it is claimed that deriving software architecture and systems from the relevant domain models can be very productive (Akşit, 2018) (Da Silva, 2015). The benefits of using models can be:

- Models can provide a high-level representation of the topic of interest and hide unnecessary details of software so that complexity can be managed;
- Models can enhance reusability and extensibility through model reuse and model transformations;
- Models can help in enforcing correctness through model-based verification techniques and testing; and
- Models can ease programming by generating code from models.

Challenges in applying models can be:

- Reducing the effort that is spent in defining models;
- Determining the abstractness and details of models;
- Checking consistency among models;
- Assuring the invariant properties of models;
- Generating efficient code from models.

Model-Driven Engineering (MDE) is a well-known approach which is based on models, meta-models (and meta-meta models, etc.) and model transformations (Brambilla, Cabot & Wimmer, 2012). During the last decade, there has been an increasing emphasis on MDE and as such many useful models are readily available for use.

Unfortunately, within the context of smart cities many useful models are still missing. We think that lack of models in designing smart-city applications is one of the causes of excessive programming effort needed in realizing smart-city applications.

Recently, a considerable amount of research has been carried out in data science, machine learning and decision support systems due to the availability of large amount of data and new data analysis and machine learning techniques (Pyne & et al., 2016) (Kramer, 2016). These techniques can be useful in obtaining realistic models by learning/inferring the (parts-of) models and/or their parameters.

We think that to design and implement the necessary models, meta models and model transformations the following research approach can be followed:

- Elaborate on smart-city applications and within the context of MDE, develop models required by most smart-city applications. In particular:
  - Develop models for the sensor networks layer. The models should represent sensor networks, topologies, data adaptation and interpretation modules, characteristics of sensors, etc.;
  - Develop models for the architectural layer. Models should represent various architectural styles that are required for smart-city applications. Important category of models are used for dependability purposes (See Challenge 3);
  - Develop models for the business platform layer: Models should represent various needs of businesses, such as work-flow modeling, scheduling and optimization;
- Develop models for the integrator language and technique (see Challenge 5) which is to be designed in the implementation of this proposal. This is important to integrate models with each other where necessary;
- Check consistency among the models automatically;
- Verify the invariants of models in the model instantiation phase, where possible;
- Adopt model query and pruning mechanisms to manage the complexity of the model base;
- Generate efficient code where applicable;
- Develop data gathering, analysis and machine learning techniques to learn/infer the (parts-of) models and/or their parameters.
- Define various realistic use-case scenarios which utilize the models defined at various abstraction levels, from the sensor network layer to the architecture and business platform layers and justify the effectiveness of the models defined. Accordingly demonstrate the advantages of the model-based approach with respect to the straightforward programming practices without using models.

The related research question is how to design high-level smart-city models, techniques and tools so that the these requirements can be fulfilled effectively.

Challenge 2: Designing a framework for managing and optimizing the configurations of clusters

Smart-city systems typically consists of clusters of systems (Obaidat & Nicopolitidis, 2016). Each system cluster is assumed to be a (potentially) software-intensive, operationally and managerially independent computer system interoperating with other systems to achieve the common business goals. Clusters may be formed at sensor network, architecture and business platform layers. Interoperation may occur within and between layers. Each cluster may also incorporate various computational elements such as sensors, translators, adaptors, etc. As such, we assume that distributed computing is a natural property of clusters. Both the elements of a cluster and the clusters of clusters may be required to be configured according to the needs.

To create generic infrastructures, the infrastructure must provide means to integrate/operate a variant set of applications. Optimizing configuration requires modeling the common and variable aspects of the clusters (Coplien, Hoffman & Weiss, 1998). Since smart cities have to cope with a various set of requirements, the design of configuration that satisfies the management and quality requirements is not trivial.

We think that managing and configuring the elements of clusters should not be restricted to procedural measures only (such as initializing, linking, etc.) but the overall quality of configurations must be considered as well. Unfortunately, to the best of our knowledge, for system structuring, generic MDE frameworks have not been developed so far to manage and optimize the elements of clusters or clusters of clusters according to the user-defined quality optimization criteria. To address this challenge, the following approach is proposed:

- Define models for clusters at sensor network, architecture, and business platform layers. Models must explicitly represent the common and variable aspects of the clusters.
- Define the necessary (quality-based) management operations for the clusters;
- Define models for various quality attributes (timeliness, energy, processing power, precision etc.) together with the necessary optimization criteria for configuring clusters optimally;
- Design and implement various optimization techniques with different strategies;
- Adopt different adaptor/generator mechanisms to restructure the clusters according to the computed optimal configuration scheme at compile time and/or at run-time.

- Develop data gathering, analysis and machine learning techniques to learn/infer the (parts-of) models and/or their parameters;
- Define various realistic management and configuration scenarios for the elements of clusters and for the clusters of clusters and apply the designed techniques accordingly. Measure the efficiency and effectiveness of the desired techniques with respect to manual and/or non-MDE approaches. Verify that the configurations are optimal with respect to the desired quality attributes and optimization criteria.

The related research question is how to design a generic MDE framework so that managing and optimizing the configuration of the elements of clusters and/or the clusters of clusters can be realized

Challenge 3: Designing models, methods and tools for critical infrastructures

Critical infrastructure for smart cities can be defined as "an asset or system which is essential for the maintenance of vital societal functions within a city. The damage to a critical infrastructure, its destruction or disruption by natural disasters, terrorism, criminal activity or malicious behavior, may have a significant negative impact for the security of the inhabitants of the city" (EU, 2019). It should be noted that this term is typically used at the scope of a nation, but similarly it applies also to smart cities which will heavily rely on the cyber part (software) that controls the city.

Since in this paper we aim at software intensive solutions to the identified smart-city problems, we will further exclude measures other than the software measures. Within this context, we claim that specially designed software systems can effectively support designing critical infrastructures by providing architectures with a high degree of reliability, availability, security, timeliness and correctness. Dependability is the ability of a system to deliver service that can justifiably be trusted (Sözer, 2009). This definition encompasses several quality attributes including reliability, availability, etc. Reliability is defined as continuity of correct service, whereas availability is readiness for correct service. Computer security or cybersecurity is defined as "the protection of computer systems from theft or damage to their hardware, software or electronic data, as well as from disruption or misdirection of the services they provide" (Wiki-b, 2019). Timeliness is defined as the ability of a computer/software system to complete its processing within the specified time constraints. Correctness is defined as the fulfillment of the functional (and qualitative) requirements of the system.

A critical infrastructure may demand all these quality attributes together and as such designing an effective critical infrastructure can be a very challenging task. Dependability can be provided by (1) prevention, for example by applying the right design methods, (2) removal of the faults, for example by verification and testing, and (3) by tolerance, for example by adopting fault-tolerance architectures. All these 3 approaches can be meaningful in a certain context.

We claim that domain-specific architectural styles and the associated techniques can help in designing critical infrastructures considerably. However, this may require the definition of domain-specific architectural styles for each quality attribute and integration of these styles together within the overall architecture by leveraging the intended quality attributes of each style.

In the literature, various algorithmic techniques and search-based methods (Harman & et al., 2012) have been introduced to compute the "optimal" architectural decomposition with respect to certain quality attributes. Although there has been a number of domain-specific architectural style proposals (Sözer, Tekinerdogan & Akşit, 2013), for example to provide a high degree of availability, composition of different architectural styles for various dependability purposes has not been studied in detail. Furthermore, to the best of our knowledge, there are no MDE framework proposals to design and integrate various architectural styles together for the purpose of supporting critical infrastructures and to compute the optimal trade-off among these styles based on user-defined criteria.

To address this challenge, it is considered necessary to design and implement various architectural styles for the purpose of critical infrastructures within the context of an MDE framework. The following approach is proposed for this purpose:

- Adopt an MDE approach to define architectural styles for: reliability and availability, security and timeliness;
- Within the MDE framework, define quality models for reliability/availability, security and timeliness;
- Define trade-off relationships among the quality attributes;
- Define models for multi-objective optimization criteria;
- Design and implement multi-objective optimization techniques with different strategies;
- Integrate (risk) analysis tools for the adopted quality attributes for example based on model-checking;
- Develop data gathering, analysis and machine learning techniques to learn/infer the (parts-of) models and/or their parameters;
- Define various realistic attack and failure scenarios and test the designed architectural styles individually and together. Verify the robustness of the styles according to the desired quality attributes. Check if the trade-off conditions are fulfilled at run-time. For the risk analysis, it is also possible to adapt other verification tools, for example by means of run-time verification. These techniques will be discussed under Challenge 4.

The related research question is, within an MDE framework, how to define a set of architectural styles for assuring certain quality attributes for critical infrastructures and how to compute and optimize the necessary trade-offs among these qualities.

Challenge 4: Optimizing the necessary quality attributes through system adaptation at run-time

Smart-city systems are long-living systems. When a system is deployed, it must be adapted, modified, extended and (partially) replaced while it is operational. Challenge 4 deals with the automatic adaptation of smart-city infrastructures at run-time.

There has been a large number of proposals to verify systems at run-time (Malakuti, 2011). Most of these systems adopt various specification languages and generate run-time monitors and verification modules according to the specifications defined. Although a considerable number of systems are now available for use, there has been only a few number of proposals to verify systems across multiple languages in distributed system settings (Malakuti, Akşit & Bockisch, 2011). As such most existing run-time verification systems are less suitable for verifying systems of systems.

A number of approaches has been presented for designing and implementing so-called quality aware architectures, for various purposes, for example for reducing energy consumption (Malakuti, Lohmann & Akşit, 2015). Self-adaptive or self* systems have been defined in the literature to create systems with dynamic adaptation (Kounev & et al., 2017). Most of these systems are inspired from the adaptive control theory (Kuo, 1995) and as such they adopt one or more control loops for adaptation. Recently, within the context of MDE approaches, a number of research proposals has been presented to design adaptive systems. So-called models@run-time systems adopt various feedback control mechanism to improve the system performance according to the predefined control parameters (Bennaceur & et al., 2014). However, there is hardly any MDE framework available which allows user-defined quality models, monitors the systems accordingly at run-time to check if the desired quality attributes are satisfied, and optimizes the system structure using self-adaptation mechanisms.

Data analytics, machine learning and decision support systems can be particularly useful in steering the control process of the adaptation of systems.

Challenge 4 is to design and implement various models for the purpose of defining run-time verification and self-adaptation systems within a context of an MDE framework. To this aim, the following approach is proposed:

- Define models of specification languages to express the desired properties of systems at run-time, for example, based on functions, data and/or time. Extend these properties according to the desired quality attributes if necessary (in alignment with Challenge 3);
- Define generative techniques for run-time monitors and verification automata;
- Minimize the overhead of run-time monitoring, where possible;
- Extend the run-time verification techniques over multiple languages and systems, for example by incorporating the run-time verification techniques in the integrating language, which is discussed by Challenge 5.
- Define models for architectural styles for self-adaptive systems based on feedback-control principles;
- Integrate self-adaptive architectural styles with the run-time verification techniques;
- Define means to gather the relevant data, and develop data analysis techniques. Accordingly, define machine learning/deep learning techniques as a meta-control mechanism of self-adaptive systems;
- Investigate multi-objective optimization techniques as control strategies;
- Define an experimental setting where the designed systems can be tested with various contextual parameters. Adopt statistical generators to adjust the contextual parameters at run-time. While varying the contextual parameters using realistic scenarios, monitor the systems behavior and its adaptation capabilities. Justify accordingly the designed systems if they satisfy the desired run-time requirements.

The related research question is how to design architectural styles for run-time verification and self-adaptation so that systems can preserve their run-time qualities as desired.

Challenge 5: Integrating software systems

Integrating software systems can be practically realized at system level or at programming-language level. “System integration is defined as the process of bringing together the component sub-systems into one system and ensuring that the subsystems function together as a system” (Wiki-c, 2019). There are various integration possibilities such as horizontal and vertical integration, which roughly correspond to communication-channel based integration or layered-architecture based integration, respectively. Programming-language level integration is realized at a much finer level of language modules. A typical smart-city application may require integration of many sub-systems and corresponding programming-language modules for the purpose of for example, data sensing, gathering and reasoning, run-time configuration and tuning, storing data on cloud systems, analyzing data, adopting machine learning techniques, utilizing decision support systems, business processes and work-flow schedulers, etc.

Many of the functionally specialized sub-systems and language modules/libraries may be developed independently. Integration is generally carried out using techniques like, IDL’s and stub-generators, adaptors, proxies, scripting languages, glue-code, brokers, virtual machines, dedicated libraries, internet protocols, web-services, service-oriented architectures, etc. Although these techniques help considerably, they may still require a considerable programming effort and in-depth knowledge about the systems/modules being integrated. The difficulties in using these techniques originate from the following observations:

- Some of these techniques are imperative techniques meaning that the programmer is obliged to write a considerable amount of program code to realize the integration. Techniques such as adaptors, proxies, scripting languages, glue-code, dedicated libraries, etc. fall into this category.
- Some of the techniques are applicable only at system-level and mostly require dedicated system/vendor specific solutions. Techniques such as IDL's and stub-generators, virtual machines, web-services, service-oriented architectures, etc. fall into this category. Since systems evolve continuously, the borders of integration cannot be fixed; programming-language level integration may turn out to be a system-level integration, or vice versa etc. System/vendor specific integration may also be a limiting factor in the evolution of systems.

Challenge 5 is to uniformly integrate sub-systems and programming-language modules with the following characteristics:

- Uniform integration is required both at system-level and programming-language level;
- Uniform integration is required at and between sensor network, architecture and business platform layers;
- Integration must be independent of systems and programming languages used;
- Both horizontal and vertical (meta-level) integration must be supported;
- Integration must be specified declaratively;
- Integration technique must respect encapsulation of systems and modules (integration through interfaces);
- Rich-set of integration semantics must be offered such as condition-based and query-based integration (For example, systems are coupled if certain conditions are TRUE);
- Integration must be realized within a single machine and/or across multiple machines;
- Integration must be verifiable as much as possible (For example, checking if the invariants of the integration holds);
- Both compilation-time and run-time integration must be supported (Declarative specifications of integrations are therefore necessary);
- Realistic case studies must be carried out for integrating various systems. For example, various sensor network subsystems, cloud-based systems, systems for big-data storage and analytics, machine-learning packages, systems used for business workflow scheduling, etc. can be used in a case study for integration. The effort spent for integration must be in average much less than the conventional techniques available.

The related research question is how to design high-level integration models, methods, techniques and tools so that Challenge 5 can be fulfilled as specified.

Challenge 6: Designing a smart infrastructure with a high degree of interoperability, configurability, adaptability and evolvability

Flexible language models and architectural styles ease coping with changes both at compile time and run-time. If the underlying languages and architectural styles are too rigid, the design space of the alternatives of design and run-time adaptations are too limited. Since smart-city systems are long-living systems, extending the life-time of the systems becomes then very difficult.

The concepts of separation of concerns and composition of concerns are fundamental in designing languages with a high degree of flexibility. The motivation here is to reduce complexity of software by decomposing software into manageable parts. Explicit composition operators are defined to create flexible systems. In practice, the term flexibility may refer to various quality attributes.

In this paper, we particularly focus on the following quality attributes:

- Interoperability: It must be possible to interoperate different systems together so that systems of systems architectures for smart systems can be realized.
- Configurability: It must be possible to configure the clusters of systems according to particular smart-city requirements (See also Challenge 2).
- Adaptability and evolvability: Systems must adapt to changing conditions and evolve with respect to changing requirements so that long-living smart-city systems can be designed.

Since mid-80's, object-oriented programming languages have started to dominate the practical usage of programming languages. In end-80's and begin-90's several researchers have claimed that objects, inheritance and message passing semantics as defined by object-oriented languages cannot express the separation of certain concerns adequately (Akşit & Bergmans, 1992). In particular, the concerns like synchronization, real-time, coordinated behavior, multiple interfaces, tracing and error handling are typical examples where object-oriented languages may fall in short.

Among others, three kinds of proposals have been quite significant: (a) reflective programming (Smith, 1982); (b) design patterns as a documentation of object-oriented solutions to recurring problems (Gamma, Vlissides, Johnson, Helm, 1994); (c) a new set of language abstractions. We will now elaborate on the latter: proposals for new language abstractions.

Aspect-oriented languages have been introduced to overcome limitations of object-oriented languages. Between 1990 and 2010 many aspect-oriented language proposals have been defined (Filman, Elrad, Clarke & Akşit, 2005). Gradually, aspect-oriented language features have been integrated into standard languages. As such these new abstractions have become commodity of programming practices. Other notable examples are ambient-oriented programing, reactive programming, feature-oriented programming, context-oriented programming and ontology-driven programming (Dedecker & et al., 2006) (Bainomugisha & et al., 2012) (Apel & Kästner, 2009) (Appeltauer & et al., 2010) (Pan & et al., (2012).

It is also possible that some states of a program may "emerge" through the interactions of software modules. These states belong to so-called "emergent behavior", which is defined as the appearance of complex behavior out of multiplicity of relatively simple interactions (Malakuti & Akşit, 2015) (Malakuti & Akşit, 2014). In general, emergent modules are created dynamically, when their creation conditions become TRUE. This requires, however, the specification of emergent conditions explicitly. The challenge is whether such conditions can be inferred and /or learned and accordingly appropriate modules are generated through the use of automatic machine learning and synthesis techniques during program execution.

Despite all these developments, providing language mechanisms and architectural styles that fulfil the flexibility needs of smart-city applications has not been accomplished yet (Sugihara & Gupta, 2008). Moreover, it is not practical to introduce a new language when so many languages are available for use. New language proposals, therefore, must extend the existing languages instead of offering a completely new language semantics and syntax.

Challenge 6 is to design and implement an extension mechanism to existing languages and/or systems so that infrastructures can be designed for smart cities with a high degree of interoperability, configurability, adaptability and evolvability, with the following characteristics:

- The extension mechanism must be usable with different languages and system implementations. From this perspective, this mechanism must be unified by the generic integration mechanism as demanded by Requirement 5;
- Explicit models must be defined for the quality attributes interoperability, configurability, adaptability and evolvability and the proposed extension mechanisms must be justified accordingly;
- Declarative extension mechanisms must be adopted instead of imperative ones (A similar condition was defined for Requirement 5);
- The extension mechanism must support various interaction modalities among modules and systems such as call-based, event-based, etc.
- The extension mechanism must be able to deal with emergent behavior, where necessary.
- Define various realistic change-case scenarios to test each quality attribute (interoperability, configurability, adaptability and evolvability) individually and together. Define typical scenarios where emergent behavior appears and disappears. Test the flexibility of the system to cope with emergent behavior. Justify that the extension mechanism works with various popular languages and systems.

The related research question is how to design extension mechanisms and architectural styles for enhancing interoperability, configurability, adaptability and evolvability of smart system applications and infrastructures.

CONCLUSIONS

This paper has first emphasized the importance of smart-city systems and then described the role of infrastructures in cost-effective realization of smart-city systems. Along this line, 6 categories of challenges are identified as possible obstacles. For each obstacle, a possible research approach is suggested. To the best of our knowledge, this is the first paper that presents these obstacles as a whole in realizing smart city systems. To address these challenges, currently we are in the process of establishing a research center at the premises of the TOBB-ET University in Ankara.

REFERENCES

Akşit, M. & Bergmans, L. (1992). *Obstacles in Object-Oriented Software Development*, in OOPSLA'92 Proceedings, ACM SIGPLAN Notices, 27 (10). pp. 341-358.

Akşit, M. (2018). *The Role of Computer Science and Software Technology in Organizing Universities for Industry 4.0 and Beyond*, Proceedings of the Federated Conference on Computer Science and Information Systems pp. 5–11.

Apel, S. & Kästner, C. (2009). *An Overview of Feature-Oriented Software Development*, Journal of Object Technology, Volume 8 Issue 5 Pages 49-84.

Appeltauer, M. & et al. (2010). *Event-Specific Software Composition in Context-Oriented Programming*, Software Composition, pp. 50-65.

Bainomugisha, E. & et al. (2012). *A Survey on Reactive Programming*, ACM Computing Surveys (CSUR).

Bennaceur A. & et al. (2014). *Mechanisms for Leveraging Models at Runtime in Self-adaptive Software*. In: Bencomo N., France R., Cheng B.H.C., Aßmann U. (Eds), Models@run.time, Lecture Notes in Computer Science, vol 8378. Springer, (2014).

Bibri, S. E. (2018). *Smart Sustainable Cities of the Future The Untapped Potential of Big Data Analytics and Context-Aware Computing for Advancing Sustainability*, The Urban Book Series, Springer.

Birolini, A. (2017). *Reliability Engineering Theory and Practice*, 8th Edition, Springer.

Brambilla, M., Cabot, J. & Wimmer, M. (2012). *Model-Driven Software Engineering in Practice*, Morgan & Claypool Publishers.

Coplien, J., Hoffman, D. & Weiss, D. (1998). Commonality and Variability in Software Engineering, IEEE Software, pp. IEEE Software 37 – 45.

Da Silva, A. R. (2015). *Model-Driven Engineering: A Survey Supported by the Unified Conceptual Model*, Elsevier Computer Languages, Systems and Structures, 43, pp. 139-155.

de Lemos, R., Gacek, C. & Romanovsky, A. (Eds) (2003). *Architecting Dependable Systems State-of-the-art Survey*, LNCS 2677, Springer.

Dedecker, J. & et al. (2006). *Ambient-Oriented Programming in Ambienttalk*, European Conference on Object-Oriented Programming, pp. 230-254.

Dhara, K. M., Dharmala, M. & Sharma, C. K. (2015). *A Survey Paper on Service Oriented Architecture Approach and Modern Web Services*, All Capstone Projects, http://opus.govst.edu/capstones/157.

DoD (2008). Systems Engineering Guide for Systems of Systems, version 1.0.

Ellis C. A. & Nutt G. J. (1980). *Office Information Systems and Computer Science*, ACM Comput. Surv. 12(1): 27-60.

EU report (2007). Smart cities Ranking of European medium-sized cities, Final report.

EU report on Critical Infrastructure (2019). Adapted from https://ec.europa.eu/home-affairs/what-we-do/policies/crisis-and-terrorism/critical-infrastructure_en

Ferrari, G. (Ed.) (2010). *Sensor Networks Where Theory Meets Practice*, Springer.

Filman, R. E., Elrad, T., Clarke, S. & Akşit M. (2005). *Aspect-Oriented Software Development*, Addison-Wesley.

Gamma, E., Vlissides, J., Johnson, R. & Helm, R. (1994). *Design Patterns Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Gil-Garcia, J. R., Pardo, T. A. & Nam, T. (2015). What makes a city smart? Identifying core components and proposing an integrative and comprehensive conceptualization, *Information Polity*, 20 pp. 61–87.

Goodfellow & et al. (2016). *Deep Learning*, MIT Press.

Gritzalis, D., Theocharodou, M. & Stergiopoulus, G. (Eds) (2019). *Critical Infrastructure Security and Resilience Theories Methods, Tools and Technologies*, Advanced Sciences and Technologies for Security Applications, Springer.

Harman, M. & et al. (2012). *Search-Based Software Engineering: Trends, Techniques and Applications*, ACM Computing Surveys, vol. 45, Issue 1, Article no. 11.

Kounev & et al. (2017). *Self-Aware Computing Systems*, Springer.

Kramer, O. (2016). *Machine Learning for Evolution Strategies*, Studies in Big Data 20, Springer.

Kuo, B. C. (1995). *Automatic Control Systems*, Prentice-Hall Inc.

Lehto, M. & Neittaanmäki P. (Eds.) (2015). *Cyber Security: Analytics, Technology and Automation*, Springer.

Malakuti, S. & Akşit, M. (2014). *Emergent Gummy Modules: Modular Representation of Emergent Behavior*, in Proc. of the 2014 International Conference on Generative Programming: Concepts and Experiences (GPCE), pp. 15-24.

Malakuti, S. & Akşit, M. (2015). *On Liberating Programs from the Von Neumann Architecture via Event-based Modularization*, in Companion Proc. of the 14th International Conference on Modularity, New York: Association for Computing Machinery (ACM), pp. 31-34.

Malakuti, S. (2011). *Event Composition Model: Achieving Naturalness in Run-Time Enforcement*, Ph.D. Thesis, University of Twente.

Malakuti, S., Akşit, M. & Bockisch, C. (2011). *Runtime Verification in Distributed Computing*, Journal of Convergence, 2 (1). pp. 1-10.

Malakuti, S., Lohmann, W. & Akşit, M. (Eds) (2015). *Introduction to Special Issue on Software Engineering Aspects of Green Computing*, In : Sustainable computing. 7, pp. 1-11.

Marinescu, D. C. (2017). *Cloud Computing: Theory and Practice*, Morgan Kaufmann.

Mohanty & et al. (2015), *Big Data A Primer*, Springer.

Nielsen, B. (2014). *Towards a Method for Combined Model-based Testing and Analysis*, in Proc. of the 2nd International Conference on Model-Driven Engineering and Software Development, pp. 609-618.

Obaidat, M. S. & Nicopolitidis, and P. (2016). *Smart Cities and Homes*, Key Enabling Technologies, Elsevier.

Pan, J. Z. & et al. (2012). *Ontology-Driven Software Development*, Springer Science & Business Media.

Panos, S. T. R. & Pardalos, M. (Eds) (2017). *Smart-city Networks Through the Internet of Things*, Springer Optimization and Its Applications 125.

Pyne, S. & et al. (Eds) (2016). *Big Data Analytics Methods and Applications*, Springer.

Sanseverino, E. R., Sanseverino, R. R. & Vaccaro, V. (Eds) (2018). *Smart Cities Atlas Western and Eastern Intelligent Communities*, Springer Tracs in Civil Engineering.

Shishkov, B. (Eds) (2018). *Business Modelling and Software Design*, 8th. International Symposium BMSD 2018, Vienna, Austria, July 2018 Proceedings, Springer LNBIP 319.

Smart Cities Council, Smart City Examples (2019). https://smartcitiescouncil.com/smart-cities-information-center/smart-city-examples.

Smith, B. C. (1982). *Procedural Reflection in Programming Languages*, Department of Electrical Engineering and Computer Science, MIT, PhD dissertation.

Sommerville, I. (2016). *Software Engineering*, 10th Edition, Pearson.

Sözer, H. (2009). *Architecting Fault-tolerant Systems*, Ph.D. Thesis, University of Twente.

Sözer, H., Tekinerdogan, B. & Akşit, M. (2013). *Optimizing Decomposition of Software Architecture for Local Recovery*, Software Quality Journal, 21 (2), pp. 203-240.

Suh, S. C. & et al. (2014) *Applied Cyber-Physical Systems*, Springer.

Sugihara, R. & Gupta, R. K. (2008). *Programming Models for Sensor networks: A Survey*, ACM Trans. Sens. Netw. 4, 2, 29 pages.

van der Linden, F. & et al. (2007). Software Product Lines in Action, Springer.

Wiki-a (2019). https://en.wikipedia.org/wiki/Smart_city Ref: 10, May, 2019

Wiki-b (2019). See for example https://en.wikipedia. org/wiki/Computer_security

Wiki-c (2019). See for example https://en.wikipedia.org /wiki/System_integration