Towards Model-Driven Simulation of the Internet of Things

Mihal Brumbulli and Emmanuel Gaudin

Abstract The Internet of Things (IoT) refers to the networked interconnection of objects equipped with ubiquitous intelligence, or simply "smart objects". The "smart" part is often followed by words like grid, home, parking, etc., to identify the application domain, and it is provided by software applications and/or services running on top of these large-scale distributed communication infrastructures. Heterogeneity and distribution scale speak for the complexity of such systems and call for a careful analysis prior to any deployment on target environments. In this paper we introduce a model-driven approach for the analysis of IoT applications via simulation. Standard modeling languages, code generation, and network simulation and visualization are combined into an integrated development environment for rapid and automated analysis.

Keywords Modeling · Deployment · Simulation · IoT · SDL · ns-3

1 Introduction

The Internet of Things (IoT) refers to the networked interconnection of billions of "smart objects", i.e., autonomous networked devices equipped with sensors, actuators, computational and storage facilities, that cooperate with each-other and the environment. In recent years IoT has gained much attention from industry and researchers with a variety of applications, e.g., smart-grid [28], smart-home [32], smart-parking [13], earthquake early warning [10], etc. This trend is sure to continue in the immediate future [11], as it continues to be supported by the

M. Brumbulli (✉) · E. Gaudin
PragmaDev, 18 rue des Tournelles, 75004 Paris, France
e-mail: mihal.brumbulli@pragmadev.com

E. Gaudin
e-mail: emmanuel.gaudin@pragmadev.com

M.-A. Cardin et al. (eds.), *Complex Systems Design & Management Asia*,
Advances in Intelligent Systems and Computing 426,
DOI 10.1007/978-3-319-29643-2_2

standardization efforts of well known organizations like ITU-T [20], ETSI [9], IEEE and IETF [16, 17].

Heterogeneity and distribution scale speak for the complexity of IoT applications and call for a careful analysis prior to any deployment on target environments. However, a thorough analysis is possible if the application is already deployed, which contradicts the purpose of the analysis in the first place. A controlled environment (e.g., experimentation test-bed [14]) can be a solution, with the advantage of having the application itself (i.e., the application intended for deployment) under analysis. Nevertheless, scalability of such environments maybe unfeasible, thus hindering the analysis of properties that can be inferred from the interaction of devices deployed in a large-scale. Simulation can address this issue, but the development of an accurate simulation model is not a trivial task. Indeed, a simulation model is an abstraction of the application (i.e., a selection of properties that are relevant to the analysis) and not the application itself. Derivation of this abstraction may become a development process in its own based on the complexity of the system, and validation is a must to ensure that it correctly represents the application intended for deployment.

In Sect. 2 we give an overview of related work on simulation of IoT applications and position our approach in respect to existing state of the art. We introduce our modeling approach by means of a typical application example in Sect. 3. The automatic transformation from model to simulation executable is described in Sect. 4. Finally, we present some conclusions in Sect. 5.

2 Related Work

We identify two types of simulators for IoT applications: specialized and generic. Specialized simulators are provided by IoT operating systems as part of their development environment, e.g., Cooja [27] for Contiki OS [8] and TOSSIM [25] for TinyOS [24]. They simulate the entire device, i.e., hardware, communication, operating system, and application. A major advantage of these simulators is that they do not require a simulation model of the application, i.e., the same description (usually code) can be used for simulation and deployment. However, the number of device models is limited to those supported by the operating system, and scalability is a possible issue as entire devices are simulated. On the other hand, generic simulators (e.g., ns-2 [2], ns-3 [15], and OMNeT++ [34]) can be used to analyze large-scale application-specific properties without the additional overhead of low-level models (e.g., hardware and operating systems), but they require a simulation model of the application.

Weingärtner et al. in [35] discuss possible solutions to the limitations of available simulators. They identify the extension of generic simulators with capabilities of executing real applications or hybrid frameworks as smarter choices.

Although extending generic simulators is quite common, it focuses on models of the underlying communication protocols and not on applications. Tazaki et al. in

[33] describe how to execute nearly unmodified applications in the context of ns-3 simulations. The aim is to increase the number of available protocol models and realism of simulations. The framework is not specifically targeted to IoT applications, however it is an important extension to generic simulation.

Brambilla et al. in [1] present a hybrid framework for the simulation of large-scale IoT scenarios. The framework is characterized by high modularity and loose coupling between simulation models to allow reuse of the code. Although it is possible to deploy the source code on real devices, this can be achieved only by adopting the required interfaces.

We follow a different approach to application development by exploiting the model-driven development (MDD) paradigm. With a pragmatic MDD approach it is possible to construct a model of the application that can be transformed into the application for deployment and/or a simulation model for analysis [31]. We use the standard languages SDL [19] or SDL-RT [30][1] in our integrated development environment[2] to capture architectural, behavioral, communication, and deployment aspects of the application into a single description (model). Automatic code generation can then transform such description into an executable simulation model for the ns-3 network simulator.

3 Modeling

Several efforts have been made in the last decade to bring together standard modeling languages with generic simulation frameworks. The advantages are obvious: while the former allow description of aspects at a higher level of abstraction independent from the target platform, the later are used to simulate such description in large-scale scenarios. Representative examples are the use of SDL with ns-2 [23] or ns-3 [3], and UML with OMNeT++ [7]. There is however an important difference between SDL- and UML-based approaches. While SDL models are used also for deployment (e.g., [10, 22]), this is not the case with UML, where models are used only for simulation.

3.1 Architecture and Behavior

In SDL the overall design is called the *system*, and everything outside of it is defined as the *environment*. The system can be composed of *agents* and *communication constructs*. There are two kinds of agents: *blocks* and *processes*. Blocks can be composed of other agents and communication constructs. When the system

[1] A pragmatic combination of the standard languages SDL, UML [26], C/C++.

[2] Real Time Developer Studio—http://www.pragmadev.com.

is decomposed down to the simplest block, the way the block fulfills its functionality is described with processes. A process provides this functionality via extended finite state machines. It has an implicit queue for *signals*. A signal has a name and parameters; they go through *channels* that connect agents and end up in the processes implicit queues. Figure 1 illustrates these concepts in a typical IoT sensor-gateway application example for parking lots.

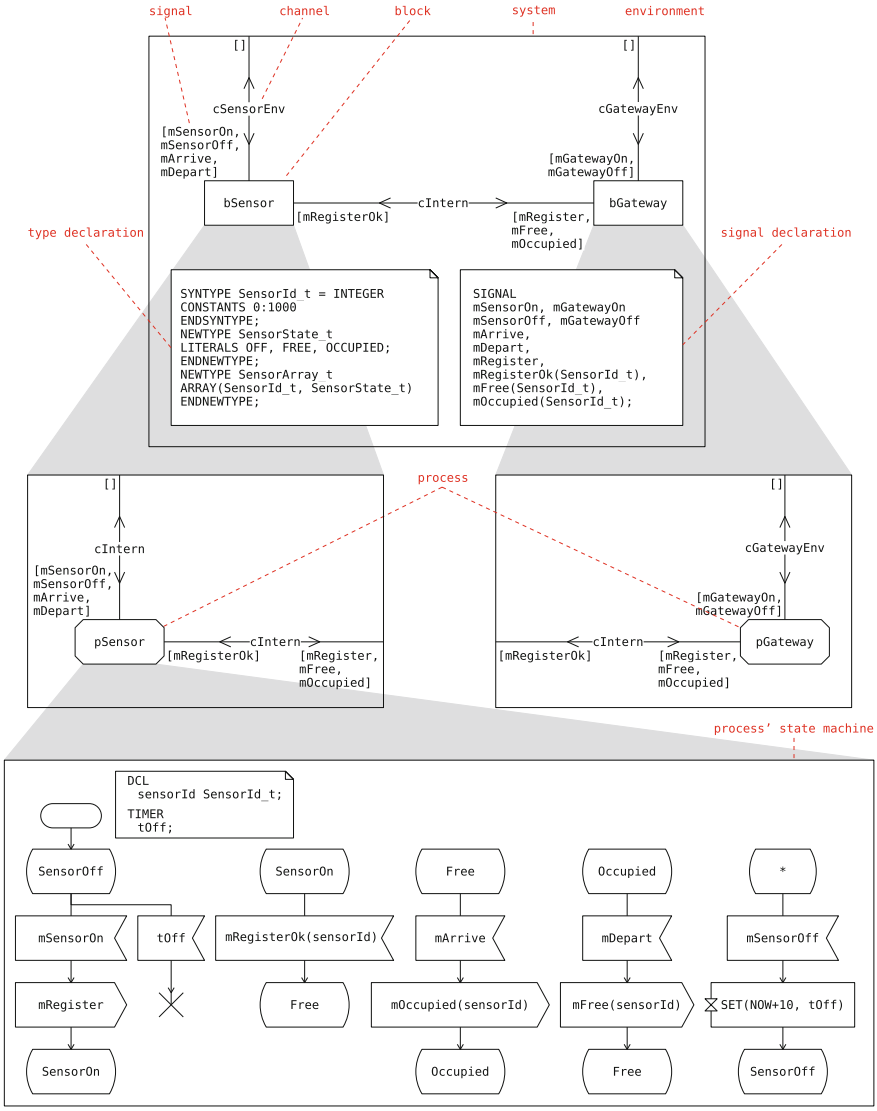


Fig. 1 Architecture and behavior description of a simple sensor-gateway application for parking lots

The gateway keeps an updated list of all sensors it is responsible for. This binding is realized at startup via the signals `mRequest` and `mRequestOk`. To keep track of the state of each individual sensor, the gateway assigns a unique identifier to it. At start, every registered sensor is in `Free` state. When a sensor detects the arrival of a car at its slot (i.e., the `mArrive` signal), it will change the state to `Occupied` and let the gateway know about such change by means of the mOccupied signal. This signal has the sensor's unique identifier as parameter, so that it is possible for the gateway to correctly update its list of available slots. When the car leaves the parking slot (i.e., the `mDepart` signal), the sensor will reset its state to `Free` and notify the gateway with the `mFree` signal. Additional signals may trigger the gateway to report the current status of the parking lot, however for simplicity these signals are not shown in the figure.

3.2 Communication

The strong point of SDL is that it allows the description of communication systems in a formal and abstract way (i.e., platform independent). However, this level of abstraction presents a number of challenges when implementation is concerned (i.e., platform specific). In this context, an important aspect to be considered is communication, which puts the internet into the internet-of-things. Communication in SDL is realized using channels, e.g., clntern in Fig. 1, and it can be local or distributed. Not making the difference between these two types of communication is an important feature of SDL that is needed to abstract from the platform. Indeed, there is no way to tell whether the pSensor and pGateway processes in Fig. 1 are running on the same node or device (i.e., local communication) or on different nodes (i.e., distributed communication). Platform specific implementation for local communication can be derived in general without much effort. The information contained in the SDL architecture and behavior description is enough to uniquely identify the sender and receiver of a signal, because all process instances are part of the same executable running on a single node. Problems begin to emerge when processes are distributed and use platform specific inter-process communication. To uniquely identify a process instance in a distributed IoT infrastructure, information about the node where the instance has been deployed is required; a typical example would be the IP address of the IoT node. However, this information is not present in a SDL model, due also to the missing concept of the node in the language.

Schaible and Gotzhein in [29] define a set of SDL patterns for describing distributed communication. The advantage of these patterns is that they are formalized, thus they can be used in every SDL description without affecting this important feature of the language. However, the introduction of patterns implies changes to an existing SDL description, and what is more important, it does not follow the choice of SDL to abstract the type of communication.

Brumbulli and Fischer in [5] apply the same concept but in the context of SDL-RT. Although not formalized,[3] the patterns are very compact, descriptive, easy to apply, and exploit the pragmatic nature of the language. Nevertheless, the problems of a pattern-based approach are still present, and it is not possible to use the patterns in SDL.

To address these issues we decided to not define and/or apply any pattern to SDL descriptions. The additional information that is required to derive platform specific implementation can be provided using SDL-RT deployment diagrams [30]. This approach does not introduce any changes to the SDL model of a system, thus keeping the desired level of abstraction in the description of communication. Also, by keeping deployment aspects (e.g., nodes and their IP addresses) separate from architecture and behavior, it is possible to use the approach for both SDL and SDL-RT. Furthermore, extending our integrated development environment with the presented approach is an important step towards a complete model-driven solution for the development of IoT applications.

3.3 Deployment

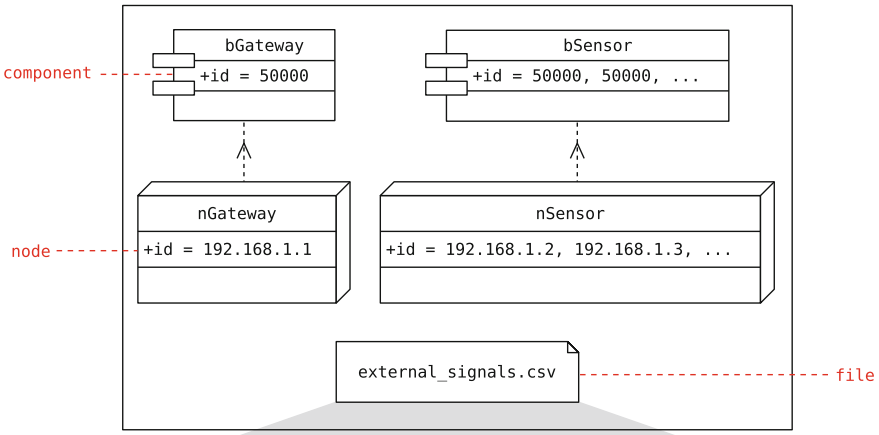
The deployment diagram describes the physical configuration of run-time processing elements of a distributed system and may contain (Fig. 2):

- *nodes* are physical objects that represent processing resources,
- *components* represent distributable pieces of implementation of a system (i.e., SDL block or process),
- *files* provide information about external signals required for simulation.

The node and component are identified by means of the id attribute. The type of values for this attribute depends on the parameters required for inter-process communication, e.g., IP address for the node and TCP port for the component. A comma separated list of values can be assigned to the attribute. This allows the description of large-scale scenarios while keeping the readability of the diagrams. The pair of id attributes is used to uniquely identify each component. The semantics of communication between two process instances are as follows:

- if the sender and receiver instances belong to the same component, then SDL communication semantics apply;
- if the sender and receiver instances belong to different components, then:
  - at first, the pair of identifiers is used to send the signal to the peer component via inter-process communication;
  - afterwards, the signal is delivered to the receiver inside the component using SDL semantics.

[3]This does not change anything in the language, because SDL-RT is not formal.



TIME	192.168.1.1:50000	192.168.1.2:50000	192.168.1.3:50000
1000	mGatewayOn		
2000		mSensorOn	mSensorOn
3000		mArrive	
4000			mArrive
5000		mDepart	
6000		mArrive	mDepart
7000			mArrive
8000		mDepart	
9000			mDepart
10000	mGatewayOff	mSensorOff	mSensorOff

Fig. 2 Deployment scenario for the sensor-gateway application for parking lots

The use of deployment diagrams was introduced [5]. We improve the approach in two significant ways:

- introduce deployment information *exclusively* in the deployment diagram, without mixing architecture, behavior, and deployment aspects of the system, thus supporting both SDL-RT and standard SDL;
- provide simple means to describe interaction with the environment.

Interaction with the environment is an important aspect that must be considered during simulation. An example of such interaction is the detection of arrival and departure of a car into a parking slot. This is modeled with the mArrive and mDepart signals in Fig. 1. These signals are addressed to the sensor, and it makes perfect sense when the system has a single sensor. However, this is not the case in our example scenario (or every IoT application scenario in general), which is composed of several sensors. In this context, external signals may be addressed to one or more distributed process instances, and to add to the complexity, they can have different parameters, timing, and order of arrival based on the intended receiver. We define a simple yet complete method for extending the deployment model with the required information. The set of external signals with parameters, timing, and receiver is described in a tabular form (i.e., comma separated values

format) in a file symbol attached to the deployment diagram as shown in Fig. 2. Each signal is configured with the time it is sent by the environment (row head) and the receiving component (pair of identifiers in column head).

4 Simulation

The SDL (or SDL-RT) description (architecture and behavior) combined with the deployment diagram are used as a basis for the generation of an executable simulation model for ns-3. A generic model for code generation was introduced in [5]. We extend the model as shown in Fig. 3 by introducing the implementation of deployment and distributed communication concepts, i.e., RTDS_DeplNode, RTDS_DeplComponent, and RTDS_Proxy.

The RTDS_DeplNode maintains the list of all its attached components in its attribute named componentsList. The RTDS_DeplComponent keeps a reference to the scheduler (RTDS_Scheduler), which manages process instances (RTDS_Proc), communication via signals (RTDS_MessageHeader), and timers (RTDS_TimerState). Local communication between process instances is handled via a memory shared mechanism. This is possible because local communication implies sender and receiver instances managed by the same scheduler. The information about the sender and receiver of a signal is encapsulated in

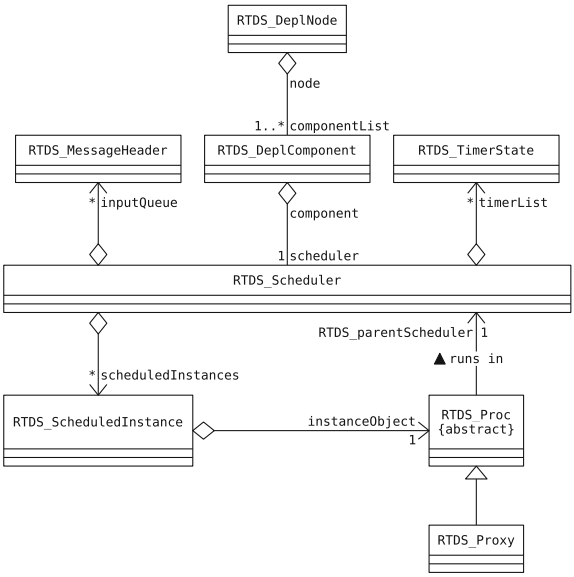


Fig. 3 Extended class diagram for code generation based on [5]

RTDS_MessageHeader. If the sender and receiver are not managed by same scheduler, then distributed communication is implied, and the signal is forwarded to the RTDS_Proxy instance. Every scheduler manages an implicit proxy instance, which interfaces to the underlying communication models provided by ns-3.

Our integrated development environment (RTDS) will automatically:

- check the syntax/semantics of the SDL description,
- check the syntax/semantics of the deployment description,
- check the syntax/semantics of the external signals given in tabular form,
- produce an executable by generating the code, compiling it, and linking it to the ns-3 library, and
- launch the executable in the background, interpret and visualize simulation events traced during execution.

Visualization of simulation events is realized as described in [4], but extended with two modes:

- in *live* mode events are visualized as they are traced from the simulation running in the background,
- in *post-mortem* mode the simulation can be replayed entirely after it has successfully terminated, which allows stepping through the events without having to re-run the simulation.

Figure 4 shows the deployment of one gateway and 99 sensors (nGateway and nSensor in Fig. 2). Nodes, their state, and distributed communication (between nodes) are visualized in the *RTDS Deployment Simulator* as shown in Fig. 4a. Nodes are displayed as colored rectangles, where the color represents the current state of the node. A sent signal is a directed arrow from the sender to the receiver, and its color represents the type of the signal. The arrow is removed from the view when the signal is received. The states and signals can be configured via the provided interface. It is possible to change their color and choose whether to display them during simulation. This feature is useful in cases where not all the states and/or signals are relevant to the analysis.

If an issue is detected in the behavior of the application (e.g., a missed state change or a signal not sent), it is possible to analyze the cause in detail by visualizing the internals of each node. This can be done on-demand using standard MSCs [18] as shown in Fig. 4b, c. MSCs are linked to the SDL description, thus identifying the source of misbehavior in a MSC trace would allow navigation to the corresponding part in the description where changes can be made.

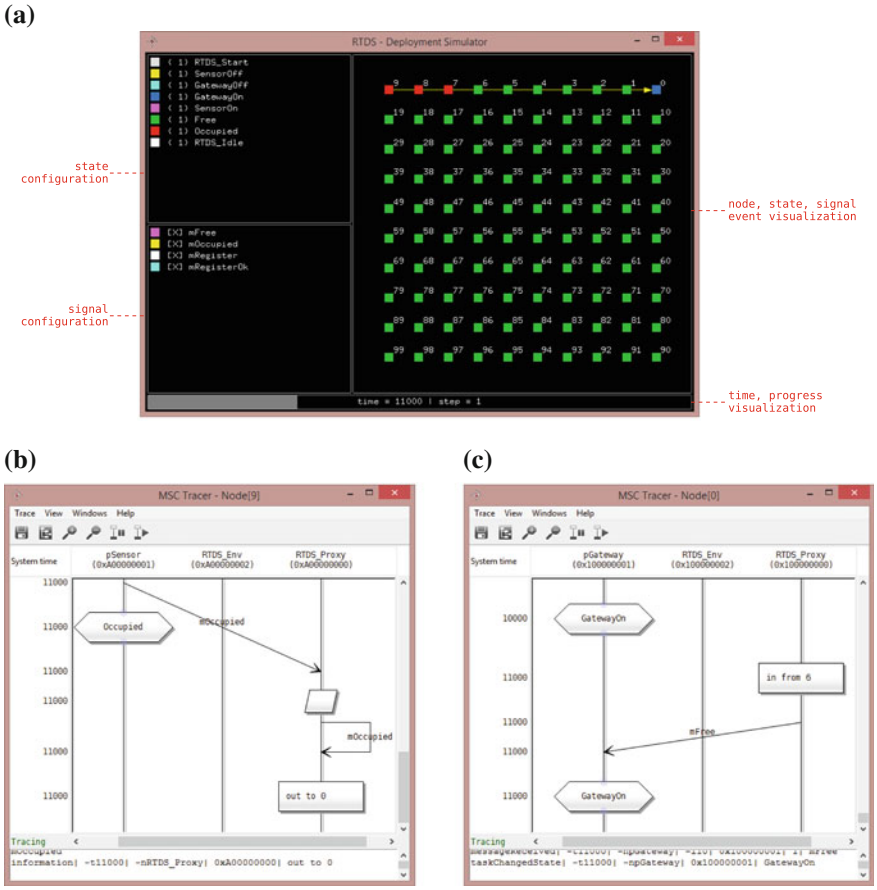


Fig. 4 Visualization of simulation events for the sensor-gateway example

5 Conclusions

In recent years IoT has gained much attention from industry and researchers, a trend that will continue in the immediate future. Heterogeneity and distribution scale contribute to the complexity of IoT applications, which need a careful analysis prior to any deployment.

In this paper we introduced a model-driven approach for the analysis of IoT applications via simulation. The standard language SDL captures architectural, behavioral, and communication aspects into a model of the application that is independent from the target platform. Deployment diagrams describe the distribution of the SDL system (blocks and/or processes) in an IoT infrastructure and the interaction with the environment. The automatic code generation transforms the description into an executable simulation model for the ns-3 network simulator.

The *RTDS Deployment Simulator* provides a graphical interface for the visualization of simulation traces.

The concepts and tools presented in this paper are important steps towards the model-driven analysis of IoT applications via simulation. In this context, the use of standard MSCs for representing traced events can aid the formal verification of properties [12]. As to how this can be applied for applications deployed in large-scale is yet to be investigated in future work. Furthermore, we are considering the possibility to extend the approach with testing support by means of standard TTCN-3 [21]. A deployment model analysis based on symbolic resolution tools such as [6] can be the next step towards a complete and fully automated approach.

References

1. Brambilla, G., Picone, M., Cirani, S., Amoretti, M., Zanichelli, F.: A Simulation Platform for Large-scale Internet of Things Scenarios in Urban Environments. In: Kawsar, F., Blanke, U., Mashhadi, A.J., Altakrouri, B. (eds.) The First International Conference on IoT in Urban Space. pp. 50–55. Urb-IoT '14, ICST (2014)
2. Breslau, L., Estrin, D., Fall, K.R., Floyd, S., Heidemann, J.S., Helmy, A., Huang, P., McCanne, S., Varadhan, K., Xu, Y., Yu, H.: Advances in network simulation. IEEE Comput. **33**(5), 59–67 (2000)
3. Brumbulli, M., Fischer, J.: SDL Code Generation for Network Simulators. In: Kraemer, F., Herrmann, P. (eds.) System Analysis and Modeling: About Models. Lecture Notes in Computer Science, vol. 6598, pp. 144–155. Springer, Berlin/ Heidelberg (2011)
4. Brumbulli, M., Fischer, J.: Simulation Visualization of Distributed Communication Systems. In: Rose, O., Uhrmacher, A.M. (eds.) Proceedings of the 2012 Winter Simulation Conferenc, pp. 248:1–248:12. WSC '12, IEEE (2012)
5. Brumbulli, M., Fischer, J.: Simulation Configuration Modeling of Distributed Communication Systems. In: Haugen, Ø., Reed, R., Gotzhein, R. (eds.) System Analysis and Modeling: Theory and Practice. Lecture Notes in Computer Science, vol. 7744, pp. 198–211. Springer, Berlin Heidelberg (2013)
6. Deltour, J., Faivre, A., Gaudin, E., Lapitre, A.: Model-Based Testing: An Approach with SDL/RTDS and DIVERSITY. In: Amyot, D., Fonseca i Casas, P., Mussbacher, G. (eds.) System Analysis and Modeling: Models and Reusability, Lecture Notes in Computer Science, vol. 8769, pp. 198–206. Springer International Publishing (2014)
7. Dietrich, I., Dressler, F., Schmitt, V., German, R.: Syntony: Network Protocol Simulation Based on Standard-Conform UML 2 Models. In: Glynn, P. (ed.) Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools. pp. 21:1–21:11. ValueTools '07, ICST, Brussels, Belgium (2007)
8. Dunkels, A., Grönvall, B., Voigt, T.: Contiki: A Lightweight and Flexible Operating System for Tiny Networked Sensors. In: Proceedings of 29th Annual IEEE International Conference on Local Computer Networks. pp. 455–462. LCN '04. IEEE Computer Society (2004)
9. ETSI: Machine-to-Machine communications (M2M); Functional architecture. ETSI Technical Specification TS 102 690, European Telecommunications Standards Institute. http://www.etsi.org/deliver/etsi_ts/102600_102699/102690/02.01.01_60 (2013)
10. Fischer, J., Redlich, J.P., Zschau, J., Milkereit, C., Picozzi, M., Fleming, K., Brumbulli, M., Lichtblau, B., Eveslage, I.: A wireless mesh sensing network for early warning. J. Netw. Comput. Appl. **35**(2), 538–547 (2012)
11. Gartner Inc.: Gartner says the Internet of Things installed base will grow to 26 billion units by 2020. http://www.gartner.com/newsroom/id/2636073 (2013)

12. Gaudin, E., Brunel, E.: Property Verification with MSC. In: Khendek, F., Toeroe, M., Gherbi, A., Reed, R. (eds.) SDL 2013: Model-Driven Dependability Engineering. Lecture Notes in Computer Science, vol. 7916, pp. 19–35. Springer, Berlin Heidelberg (2013)
13. Geng, Y., Cassandras, C.: New "smart parking" system based on resource allocation and reservations. IEEE Trans. Intell. Transp. Syst. **14**(3), 1129–1139 (2013)
14. Gluhak, A., Krco, S., Nati, M., Pfisterer, D., Mitton, N., Razafindralambo, T.: A survey on facilities for experimental internet of things research. IEEE Commun. Mag. **49**(11), 58–67 (2011)
15. Henderson, T.R., Roy, S., Floyd, S., Riley, G.F.: ns-3 Project Goals. In: Proceedings of the 2006 Workshop on ns-2: The IP Network Simulator. p. 9. WNS2 '06. ACM, New York (2006)
16. IEEE: IEEE Standard for Local and metropolitan area networks—Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). IEEE Standard 802.15.4, Institute of Electrical and Electronics Engineers. http://standards.ieee.org/findstds/standard/802.15.4-2011.html (2011)
17. IETF: Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks. Standards Track RFC 6282, Internet Engineering Task Force. http://www.rfc-editor.org/info/rfc6282 (2011)
18. ITU-T: Message Sequence Chart (MSC). ITU-T Recommendation Z.120, International Telecommunication Union—Telecommunication Standardization Sector. http://www.itu.int/rec/T-REC-Z.120/en (2002)
19. ITU-T: Specification and Description Language—Overview of SDL-2010. ITU-T Recommendation Z.100, International Telecommunication Union—Telecommunication Standardization Sector. http://www.itu.int/rec/T-REC-Z.100/en (2011)
20. ITU-T: Overview of the Internet of Things. ITU-T Recommendation Y.2060, International Telecommunication Union—Telecommunication Standardization Sector. http://handle.itu.int/11.1002/1000/11559 (2012)
21. ITU-T: Testing and Test Control Notation version 3: TTCN-3 core language. ITU-T Recommendation Z.161, International Telecommunication Union—Telecommunication Standardization Sector. http://www.itu.int/rec/T-REC-Z.161/en (2014)
22. Kuhn, T., Gotzhein, R., Webel, C.: Model-Driven Development with SDL: Process, Tools, and Experiences. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) Model Driven Engineering Languages and Systems. Lecture Notes in Computer Science, vol. 4199, pp. 83–97. Springer, Berlin Heidelberg (2006)
23. Kuhn, T., Geraldy, A., Gotzhein, R., Rothländer, F.: ns + SDL: The Network Simulator for SDL Systems. In: Prinz, A., Reed, R., Reed, J. (eds.) SDL 2005: Model Driven. Lecture Notes in Computer Science, vol. 3530, pp. 1166–1170. Springer, Berli (2005)
24. Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., Culler, D.: TinyOS: An Operating System for Sensor Networks. In: Weber, W., Rabaey, J., Aarts, E. (eds.) Ambient Intelligence, pp. 115–148. Springer, Berlin (2005)
25. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In: Akyildiz, I.F., Estrin, D., Culler, D.E., Srivastava, M.B. (eds.) Proceedings of the 1st International Conference on Embedded Networked Sensor Systems. pp. 126–137. SenSys '03. ACM, New York (2003)
26. OMG: OMG Unified Modeling Language (OMG UML), Superstructure. Version 2.4.1. OMG Standard, Object Management Group (2011)
27. Österlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T.: Cross-Level Sensor Network Simulation with COOJA. In: Proceedings of the 31st IEEE Conference on Local Computer Networks. pp. 641–648. LCN '06. IEEE Computer Society (2006)
28. Ramchurn, S.D., Vytelingum, P., Rogers, A., Jennings, N.R.: Putting the 'smarts' into the smart grid: a grand challenge for artificial intelligence. Commun. ACM **55**(4), 86–97 (2012)

29. Schaible, P., Gotzhein, R.: Development of Distributed Systems with SDL by Means of Formalized APIs. In: Reed, R., Reed, J. (eds.) SDL 2003: System Design. Lecture Notes in Computer Science, vol. 2708, pp. 158–158. Springer, Berlin (2003)
30. SDL-RT Consortium: Specification and Description Language—Real Time. SDL-RT Standard V2.3, SDL-RT Consortium. http://www.sdl-rt.org/standard/V2.3/html/index.htm (2013)
31. Selic, B.: The pragmatics of model-driven development. IEEE Softw. **20**(5), 19–25 (2003)
32. Silva, L.C.D., Morikawa, C., Petra, I.M.: State of the art of smart homes. Eng. Appl. Artif. Intell. **25**(7), 1313–1321 (2012)
33. Tazaki, H., Urbani, F., Mancini, E., Lacage, M., Câmara, D., Turletti, T., Dabbous, W.: Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments. In: Almeroth, K.C., Mathy, L., Papagiannaki, K., Misra, V. (eds.) Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies. pp. 217–228. CoNEXT '13. ACM (2013)
34. Varga, A., Hornig, R.: An Overview of the OMNeT ++ Simulation Environment. In: Molnár, S., Heath, J.R., Dalle, O., Wainer, G.A. (eds.) Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems. p. 60. SimuTools '08, ICST, Brussels, Belgium (2008)
35. Weingärtner, E., Ceriotti, M., Wehrle, K.: How to simulate the internet of things? In: Proceedings of the 11th GI/ITG KuVS Fachgespräch Drahtlose Sensornetze. pp. 27–28. FGSN '12. Technische Universität Darmstadt (2012)