Towards User-centric DSLs to Manage IoT Systems

Moussa Amrani, Fabian Gilson, Abdelmounaim Debieche and Vincent Englebert
University of Namur, PReCISE Research Center, Namur, Belgium

Keywords: Model-driven Engineering, Internet of Things, Domain-specific Language, Rule-based Semantics.

Abstract: Hidden behind the Internet of Things (IoT), many actors are activelly filling the market with devices and services. From this profusion of actors, a large amount of technologies and APIs, sometimes proprietary, are available, making difficult the interoperability and configuration of systems for IoT technicians. In order to define and manipulate devices deployed in domestic environments, we propose IoTDSL, a Domain-Specific Language meant to specify, assemble and describe the behaviour of interconnected devices. Relying on a high-level rule-based language, users in charge of the deployment of IoT infrastructures are able to describe and combine in a declarative manner structural configurations as well as event-based semantics for devices. This way, language users are freed from technical aspects, playing with high-level representations of devices, while the complexity of the concrete implementation is handled in a dedicated layer where high-level rules are mapped to vendor's API.

1 INTRODUCTION

Facing the explosion of available connected devices, many vendors are jumping into the market, proposing a large spectrum of products ranging from connected devices to associated end-user services (Lee and Lee, 2015). This results in a wide heterogeneity in software and hardware implementations, as well as an ever growing list of concerns and opportunities in terms of interoperability, data management, privacy and scalability (Chaqfeh and Mohamed, 2012).

As the Internet of Thigs (IoT) infiltrates many aspects of people's life through their cars, heating systems, phones and so forth, a critical challenge is to provide end-users the possibility to benefit from the plethora of connected devices and configure them for their particular needs. From the recent research conducted at the University of Namur, we identified how difficult it is to make *things* cooperate, and describe things configurations, because of the major influence of distinct technologies. In order to hide vendor-specific implementation details, we target a dedicated technology-agnostic environment to adapt and combine IoT solutions.

Model-Driven Engineering (MDE) has been recognised during the last decade as a software engineering technique dedicated to the design, management and evolution of computer languages enabling automatic generation of production code, diverse types of analysis and early verifications. Following this trend, we introduce IoTDSL, a prototype Domain-Specific Language (DSL) meant to capture IoT devices capabilities and their deployment configurations, while providing a declarative way to end-users, letting them achieve their own scenarios.

Outline. We start in Section 2 by presenting archetypal scenarios of IoT devices usage to motivate why and how it becomes important to bring end-users back in control of the devices in their own domestic environment. We then extract the main crucial IoT challenges specific to the use of DSLs and MDE techniques to realise this vision. In Section 3, we introduce IoTDSL, our prototype DSL to specify and interconnect devices in an intuitive and general way, and illustrate its use through a typical example. We overview in Section 4 the use of DSLs for IoT, comparing existing approaches with ours and assessing them against the challenges we identified; then conclude in Section 5 and present the main lines of work ahead to transform our prototype in a fully functional DSL.

2 CONTEXT & CHALLENGES

Domestic IoT devices could be used in many different situations for many purposes. Consider these two examplars for smart homes:

1. Alice lives in a house equipped with a number of devices: door and window lock detectors, devices controlling the lights, temperature and hu-

Amrani M., Gilson F., Debieche A. and Englebert V.
Towards User-centric DSLs to Manage IoT Systems.
DOI: 10.5220/0006285405690576
In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, pages 569-576
ISBN: 978-989-758-210-3

midity, security devices for detecting smoke and carbon dioxide, as well as a plethora of entertainment devices for TV, music, and sport. She is an active workwoman, so she often reconfigures her home devices to accommodate her changing lifestyle: for example during winter, she's staying inside, mainly teleworking, and needs her bathroom and living room to stay warm enough, but not too much, while during summer, she spends more time outside expecting her home to stay safe and being notified of any intrusion.

2. Bob is 68 years old and leaves alone at home, since his children have their own family lives on the other side of the city. As many elderly, Bob suffers from ageing diseases, but prefers to stay in a familiar environment rather than leaving in an dedicated institution. His house is equipped with devices monitoring potential falls that can have tragic consequences for him. It is also equipped with nowadays entertainment devices such as a smart TV and phone. He wants to feel safe at home: if he falls and cannot stand up in the bathroom, he would like to warn his family and notify daycare nurses to rescue him quickly.

These typical configurations share two commonalities. First, both houses integrate common connected devices fulfilling the same functionalities, *e.g.*, monitoring the temperature, or sending messages after the detection of an abnormal situation, but are likely different in terms of vendors, communication protocols and detailed properties. For example, a temperature sensor could be paired with a heating system to maintain a given temperature inside the house, while in other configurations, both devices are clearly separated. Second, the way these devices interact with each others highly depends on the end user, and more likely for the same user, on the actual context. On the one hand, in Alice's situation, different actions on the same devices are required throughout the year. On the other hand, Bob's situation can be replicated to another house, but with different sets of concerns related to ageing diseases.

We argue that a good way of capturing those potential variations of devices and situations is to provide to end-users, *i.e.* home inhabitants like Alice, or technicians equipping houses like Bob's one, a DSL that provides the following features:

Device Description. A precise inventory of the devices used in a specific deployment as well as the high-level capabilities of these devices, described in terms that are immediately understandable by end-users, as opposed to conveying technical details about how those devices precisely operate;

Network Description. A way to capture where each device is located and how it is possible to communicate with it, in order to receive or send data to it;

Dynamics. A way to describe the interactions wished by end-users, *i.e.* how to leverage the functionalities of the devices to effectively realise one or several scenarios that are convenient for the end-users.

Those features are obviously not sufficient to obtain a fully-fledged solution that becomes adaptable to any situation, but they still represent necessary steps to provide end-users the capacity to manipulate a collection of devices without relying on specific technologies. Though, the definition of such DSLs raises some questions directly related to the hypotheses assumed by those features, that we explore and relate to each other in the rest of this section.

Capability Discovery. Providing the ability to drive interconnected devices assumes the capacity of automatically discovering devices' capabilities in a standardised and uniform way (Chaqfeh and Mohamed, 2012). Similar processes exist for other technologies, like USB devices plugged into computers that automatically expose their natures and capabilities (*e.g.*, a pointing or video device). Classifying such capabilities could be useful to build an ontology of normalised functions that could result in powerful APIs to manipulate devices.

Complex Event Processing (CEP). Letting end-users deal with devices through their low-level capability interfaces could lead to confusion and stiff complexity for defining usage scenarios (Ma et al., 2013). Rather, providing a way of reifying low-level device computations into high-level events could help end-users leverage the complexity of devices networks and pave the way to manipulate them freely and transparently (Cugola and Margara, 2012). Since CEP consists of deriving meaningful conclusions from a stream of events occurring within a system and responding to them as quickly as possible, it provides a solution to extract meaningful events from low-level computations. However, for a solution to be complete and useful, the reverse operation should be addressed: high-level actions should be adequately translated into low-level actuations.

Protocol Interoperability. A domotic solution with heterogeneous devices would often integrate elements from various providers, thus communicating through multiple protocols. In order to make them interact efficiently, without forcing end-users to stick with one vendor that can dictate costs and restrictions, a powerful DSL should provide ways for interoperability over multiple communication protocols, without requiring end-users to understand the protocols' intricacies, versions and technical restrictions (Gubbi et al., 2013).

Scalability. As the number of application domains increases, the amount of connected devices is expected to rise exponentially. When updating existing IoT configurations, current solutions may not collapse when adding more elements (Mukhopadhyay, 2014). Furthermore, a DSL must provide a way to absorb scalability problems, hiding as much as possible purely technical constraints regarding increases in size and complexity of operating configurations.

Data Management. Analogously to scalability issues, the massive increase in connected devices will produce more and more data to be processed, stored and, for some of them, post processed (Lee and Lee, 2015). More data means seemingly more storage capabilities and the required space to handle such flow of information will be at its highest ever. Furthermore, the multiplication of available (sensors) sources is creating a whole new world of data processing and mining possibilities, but also a profusion of divergent concrete data types that sooner or later must be mapped to equivalent concepts.

Non-Functional Properties. A powerful DSL should encompass typical non-functional properties of device networks to ensure long-life and secure realisation of scenarios. *Performance* is crucial, and depends both on the devices capabilities but also on the quality of the communication network: any source of latency could have dramatic impacts that can lead to critical situations. *Resource availability*, both in terms of computation and memory capability, but also in terms of energy, is another crucial bottleneck for the adoption of DSLs as a solution for defining scenarios. The code generated from the DSL should not overload the devices with repetitive communications or unnecessary computations that would drain the device's battery. *Security* is yet another concern with respect to two aspects. First, sensitive data could be exposed through the communication network, endangering users privacy. Second, some functionalities could be locked and only accessible to authorised users (Tan and Wang, 2010).

3 IoTDSL

Building a well-calibrated DSL is known to be difficult and error-prone. It usually requires a broad expertise on a domain before a consensus emerges on which concepts are first-class citizens and how to effectively represent them. Fortunately, MDE technologies operated substantial breakthrough over the past decade, allowing language designers to define their own DSL structures and user interfaces more easily. Visual syntax is often a needed feature to any language, allegedly to simplify its understanding and manipulation. However, in order to rapidly obtain a functional prototype, we chose to start from a textual syntax, way easier to manipulate and evolve for language designers. At last, we plan to propose a visual syntax: the transition should be facilitated since our DSL is developed under *GeMoC* (Bousse et al., 2016), a MDE framework that supports both visual and textual representations as concrete syntaxes and maintains a full synchronisation between them.

Based on the challenges identified in section 2, we now introduce IoTDSL, our DSL devoted to facilitate the high-level manipulation of IoT systems. At the heart of IoTDSL are two governing principles. First, we promote a clean separation of concerns for all aspects the DSL has to handle, embedding several sublanguages. We believe this approach to be scalable, and to support independent evolutions of each concern without impacting the other aspects, since those aspects are composed through well-defined interfaces. Second, our DSL relies on events, a natural paradigm for specifying various models of interactions that is widely used in embedded and critical systems, and where a clear separation between the system and its environment is performed, further empowering the separation of concerns.

Despite its early stage of development, IoTDSL shows its ability to capture the definition of small-scale IoT systems appropriately. We first extract a simple scenario from our project at the University of Namur to demonstrate typical usages of IoT systems deployed inside a house. We then simultaneously explain each part of the definition of IoTDSL through dedicated examples.

3.1 Running Example

To illustrate our proposal, we consider a smart home equipped with several devices illustrated in Figure 1. At the entrance, the door is equipped with a lock detector, checking when the door is opened or closed. The hallway contains presence detectors, so that the lights in the hallway as well as in the living room automatically switch on when Alice gets back home. Furthermore, the living room also contains a presence detector because Alice wants the temperature to be automatically monitored when she's occupying the room, being maintained between 20°C and 22°C. Otherwise, when she is not home, the temperature

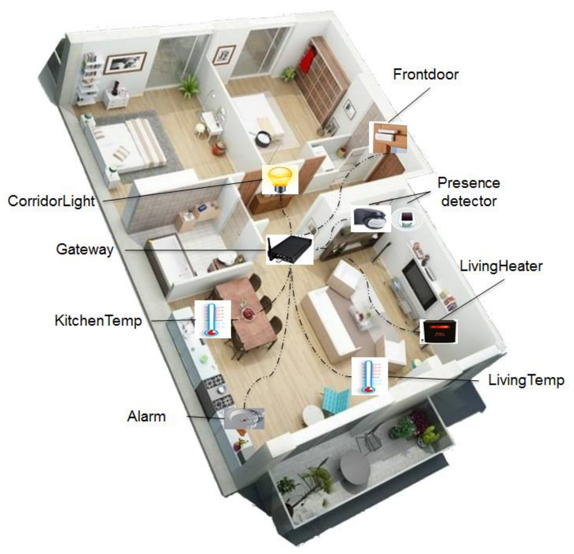

Figure 1: Alice's smart home equipped with various devices, interacting through a centralised gateway.

may not drop below 16°C. For security reasons, Alice's kitchen contains a smoke detector and a temperature monitor. When she cooks, it happens that she burns something; but she wants to make sure to detect any departing fire: she considers a critical situation when the temperature remains above 40°C consistently during five minutes while there is also smoke in the kitchen.

3.2 Type Definition

In this section, we define IoT devices' types, *i.e.* which capabilities are available to the users in terms of environment sensing and actuating. In our scenario, type definitions either come from an advanced user who is able to reason properly about a particular device and extract the relevant information, or from a preexisting devices database, either being an repository the system is connected to, or a library of *devices-off-the-shelf*.

The concepts dedicated to type definition are shown in Figure 2 (green background). This part is similar to the notion of Classifier in MOF-like languages: a Type is either a PrimitiveType, or a user-defined DeclaredType. We distinguish between general Gateways, which centralise information and processing, and Nodes deployed in the environment, having capabilities and transfering data to Gateways. A Capability is basically a MOF-like operation with a list of Parameters that either captures data from the environment, or acts on it. A Node can mix both kinds of capabilities, allowing us to represent complex behaviour in a uniform fashion.

For now, types are defined in specific files that can be imported and combined easily within IoT specifications. We plan to propose a graphical interface to facilitate the browsing and integration of library components into IoT systems.

```
gateway Central
  device DoorLock {
  sensing opened()
  sensing closed()
}
device LightBulb {
  actuating on()
  actuating off()
  actuating blink()
}
device TempSensor {
  sensing getTemp()
}
device SmokeDetector {
  sensing smoke()
}
device Heater {
  actuating warm(in delay: Int)
  actuating stop()
}
device Alarm {
  actuating sound()
}
device Timer {
  actuating set(in delay: Int)
  sensing timeout()
}
device PresenceDetector {
  sensing detected()
  actuating isPresent(out p:Bool)
}
```

Listing 1: Type declarations in IoTDSL: sensing or actuating capabilities as high-level events.

Listing 1 illustrates how devices are declared in IoTDSL. Each device is introduced by the keyword device, has a name and capabilities that correspond to reporting events (sensing) or operating over the environment (actuating). A special device, introduced by the keyword gateway, centralises data from all devices connected to it (cf. Section 3.3). Note that this is the visible part of IoTDSL: in the background, events declared for all devices need to be mapped to concrete low-level APIs events using a dedicated mapping language that is not detailed here for space reasons.

3.3 Network Configuration

The configuration-related constructs are specified in the purple-part of Figure 2. A concrete device is materialised by a NodeInstance and is typed by an abstract Node, *e.g.* a Device or a Gateway. Instances may communicate with other IoT devices through predefined CommunicationPaths. Such paths define, among others, one or more protocols used to interact. We actually rely on existing platforms, such as OpenRemote[1] or SmartThings[2] to handle the intricate details of the protocols since such details are, from an end-user point of view, technical aspects rather than essential matters. By knowing which protocols are used between each pair of devices, we can automatically perform data conversion in the proper format required by the protocols: most of those protocols are already implemented in *General-Purpose Programming Languages* (GPLS), like Java or C.

[1]http://www.openremote.org
[2]https://www.smartthings.com/

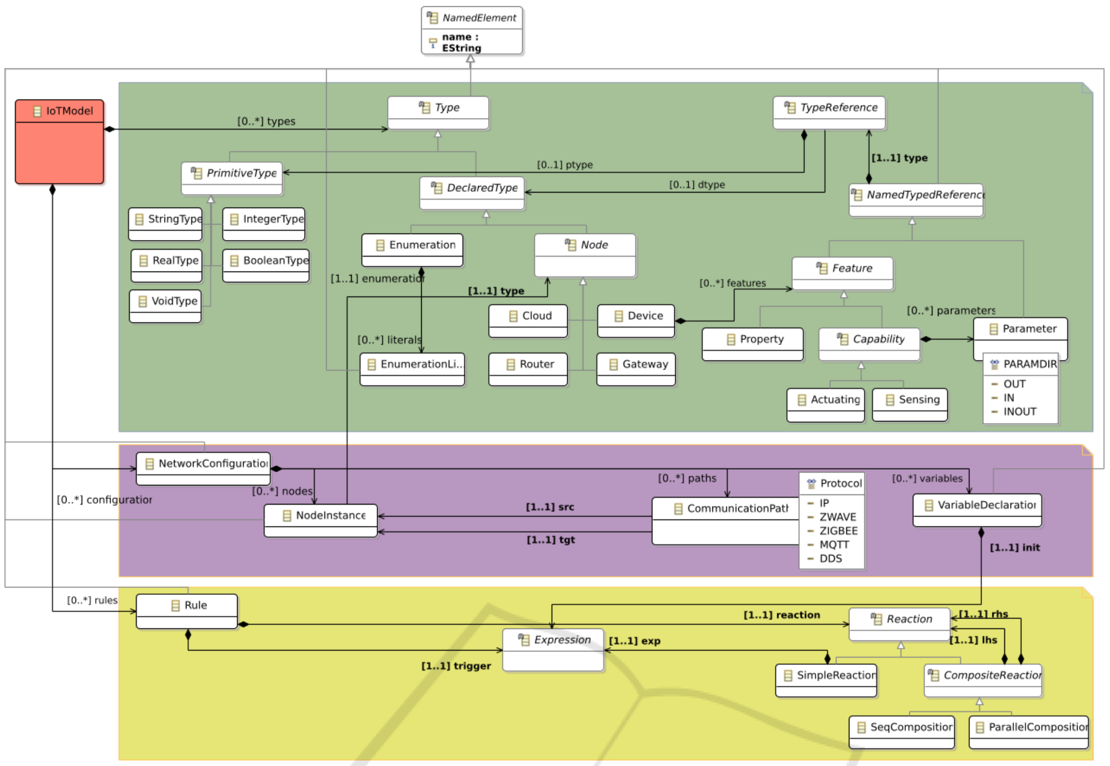

Figure 2: Metamodel of IoTDSL, separated in three concerns: *Type Definition* captures devices' capabilities (top green part), *Network Configuration* details how device instances are connected to each others (middle purple part), *Business Rules* defines the functionalities expected from the IoT installation (bottom yellow part).

Concretely, we can easily imagine that network configurations are automatically established if devices follow the recommendation of discovery protocols (Al-Fuqaha et al., 2015). By concentrating the connections on a gateway in charge of centralising joining and leaving devices, *e.g.*, phones that enter or leave a house, it becomes possible to gather information on new devices and connect them appropriately to other devices in the network according to their capabilities, communication protocols, and so on. Although IoTDSL empowers Nodes to be passed as Parameters, enabling devices' definition to be discovered thanks to the Gateway.

Listing 2 shows the connection between the devices presented in Figure 1. A specific device is considered as an instance of a defined type such that particular devices with the same set of capabilities may be distinguished via identifiable references. Communications are purely declarative and only mention the protocol type (introduced by the via keyword). Note that we added an extra instance timer : Timer used to simulate time periods. This *pseudo* device is used to artificially set a clock time, whose behaviour is directly linked to the supporting platform of the gateways for time management.

```
configuration MyHome {
  node gw              : Central
  node livingTemp      : TemperaturSensor
  node kitchenTemp     : TemperaturSensor
  node livingHeater    : Heater
  node frontdoor       : DoorLock
  node corridorLight   : LightBulb
  node livingLight     : LightBulb
  node timer           : Timer
  node alarm           : Alarm
  node corridorDetector: PresenceDetector
  node livingDetector  : PresenceDetector
  node smokeDetector   : SmokeDetector

  from livingTemperature to gw via MQTT
  from HomeTemperature to gw via DDS
  from livingHeater to gw via DDS
  from frontdoor to gw via MQTT
  from light to gw via MQTT
  from timer to gw via DDS
  from alarm to gw via DDS
  from bodydetector to gw via MQTT
  from smokeDetector to gw via MQTT
}
```

Listing 2: Network Configuration for Alice's House.

3.4 Business Rules

This last part of our DSL is the heart of IoT systems manipulation and is detailed in the bottom yellow part of Figure 2. It relies on an event-based framework consisting of a set of Rules that implement various functionalities an end-user wants to achieve. Rules' triggers are cyclically evaluated against the surrounding environment. Whenever a trigger Expression becomes true, it executes the appropriate reaction associated to the rule. A trigger is defined as an expression, whose precise definition is omitted here since it simply follows any expression language available in GPLs for navigating nodes and evaluating boolean or arithmetic expressions (note our DSL does not contain boolean negation, since detecting the *absence* of events is known in CEP as being difficult). A reaction defines a sequential or parallel combination of capabilities, enabling to sort actions by, or require data from some identifiable devices. A typical reaction may be to switch on all lights in a house, or only the ones of a certain type.

For now, our approach is purely middleware-oriented: rules are gathered into a gateway. For efficiency and resource consumption reasons, we are also exploring how to automatically identify parts of the business logics that can be exported to advanced nodes with sufficient processing and power resources, in order to lower network and gateway overuses.

We now explain how the scenario presented in the running example is translated in business rules in IoTDSL. We identified three different situations where Alice needs to specify what she expects from the devices of her house:

1. When Alice gets home (and thus opens the front door), she wants the lights in the hallway and the living room to be automatically switched on.

```
rule SwitchLightsWhenEnter:
  when (frontdoor.opened() and after
       corridorDetector.detected()) do {
    corridorLight.on() || livingLight.on()
  }
```

2. When Alice is in the living room, the temperature inside the room should be maintained between comfortable temperatures; whereas when she is not, the temperature should not drop below 16°C.

```
rule PresentInLiving:
  when (timer.timeout()) do {
     livingDetector.isPresent() || presenceTimer.
          set(TIMEDETECT)
  }
rule MonitorLivingTempInStop:
  when (livingDetector.detected() and livingTemp.
       getTemp > IN_MAX) do {
    livingHeater.stop()
  }
rule MonitorLivingTempInStart:
  when (livingDetector.detected() and livingTemp.
       getTemp < IN_MIN) do {
    livingHeater.warm(TIMEHEAT)
  }
rule MonitorLivingTempOutStart:
  when (timer.timeout() and before livingTemp.
       getTemp < OUT_MIN) do {
    livingHeater.warm(TIMEHEAT)
  }
```

3. There is a critical fire situation when smoke is detected in the kitchen while temperature is upper 40°C for at least five minutes.

```
rule AlarmWhenSmokeAndHighTemp:
  when (kitchenTemp.getTemp() > 45
         within 5 min from smokeDetector.smoke()) do
              {
     alarm.sound()
  }
```

These rules show some particularities of IoTDSL's Rule language. First, it is possible to define local or global variables and reuse them as part of parameters or expressions (cf. Rules PresentInLiving or MonitorLivingInStop, among others). Second, it is possible to compose reactions either in sequence (class SeqComposition in the metamodel of Figure 2, textually represented as «;») or in parallel (class ParallelComposition, represented as «||»), like Rule PresentInLiving demonstrates. Third, IoTDSL's expression language integrates a simple mechanism for events time management: the before and after event modifiers check that an event happens in a reasonable time window before/after the one it is combined to (in Rules SwitchLightWhenEnter and MonitorLivingTempOutStart, events are combined with and for checking quasi-simultaneity); while the within...from construct detects the occurrence of an event within a given duration after another event (cf. Rule AlarmWhenSmokeAndHighTemp). Fourth, because IoTDSL cannot check the non-occurrence of events, time management is crucial for ensuring a sound detection of interesting non-events: for example here, Rule PresentInLiving is used to periodically check for presence in the living room (with the actuation isPresent()); but when nobody is detected, we still have to maintain the temperature above OUT_MIN = 16°C. By detecting that timeout() occurred previously, we still are capable of adjusting the living room temperature adequately.

4 RELATED WORK AND DISCUSSION

A series of overviews have been recently conducted on several aspects of IoT. In (Al-Fuqaha et al., 2015; Xu et al., 2014a), the authors reviewed the applications, protocols and technologies used in the distinct IoT layers, while (Singh et al., 2014; Gubbi et al., 2013) focused on architectural aspects and (Tan and Wang, 2010; Xu et al., 2014b) reviewed security ones. Most of these contributions identify a number of challenges crossing the application domain of a DSL for IoT, from which we identified the most relevant ones to our contribution.

Capturing variations of a domain with explicit constructs close to the domain concepts resides at the essence of DSLs. In that regard, many DSLs were proposed for various purposes in the IoT stack. CHAR-

IOT (Pradhan et al., 2015) addresses Cyber-Physical Systems by providing a component model that clearly distinguishes between communication and computation, while ensuring resilience features in highly reconfigurable systems. In (Brandtzæg et al., 2012) is presented a DSL aimed at facilitating the deployment of applications, based on a component model of the environment used to locate the architecture nodes where business logic can be leveraged. ALPH (Munnelly and Clarke, 2008) is a DSL for ubiquitous healthcare that focuses on three concerns: mobility, by helping users to manage frequent devices disconnections; context-awareness to adapt application behaviour to environmental changes; and infrastructure, for managing the heterogeneity of communication protocols. Midgar (García et al., 2014) offers a visual interface to support end-users in controlling interconnected devices and generate the glue application making these devices interoperate. In (Salihbegovic et al., 2015), the authors present a visual DSL for capturing the features and intercommunications of devices distributed in various application domains spanning from smart homes to patient monitoring. These contributions target different application domains at different abstraction levels, but possess every key features we identified in Section 2 in a more or less explicit way. Since our DSL targets end-users with no prior knowledge in programming, we contrast with these contributions by having a textual representation for now, but by offering a more intuitive, declarative style for expressing the system's dynamics through semantics rules.

ThingML (Harrand et al., 2016) is the closest contribution to our DSL: it uses a similar device description with messages and communication ports attached to devices, but describes the dynamics of devices and systems through state machines, which appear to be more obscure for end-users. However, the conceptual drawbacks are similar in both paradigms: state machines need to be deterministic on their transitions, while rules have to avoid multiple concurrent firing to avoid executing several rules at the same time.

Other approaches, *e.g.* (Cheng et al., 2016; Bhandari and Bergmann, 2013), relying on the *Event Condition Action* (ECA) paradigm, share a similar view for IoT devices orchestration through CEP, though not having the same expressiveness for devices'definition.

All previous contributions take advantages of MDE technologies and tools. More general MDE framework like GeMoC (Bousse et al., 2016) or ThingML allow to specialise the description of interconnected devices, for example to describe Arduino systems specifically in ArduinoML (Mosser et al., 2014).

5 CONCLUSION AND FUTURE WORK

In this paper, we presented a prototype DSL named IoTDSL, designed for capturing the definition of devices' capabilities and their concrete deployment in specific configurations. It provides a declarative language based on rules where end-users may define their own scenarios by manipulating devices through high-level concepts. We also identified key challenges from the literature specific to DSL engineering in the area of IoT.

At the heart of IoTDSL is a clear separation of three main concerns that any DSL for IoT should address. By raising the abstraction level and offering a conceptual view of devices' capabilities, IoTDSL promotes reuse through dedicated device libraries, strongly suggesting a standardisation of interfaces like the ones already existing in other domains (for example, the many computer devices using USB). Since IoTDSL is developed with MDE tools, adding a visual syntax is almost straightforward. The communication between devices is a volatile domain, with new protocols emerging every year. By only declaring how things communicate, we push the burden of translating / extracting data from low-level protocols to high-level interfaces towards technicians in charge of defining and understanding such protocols. However, this task is done only once per protocol, and can reuse the experience and techniques already available in other areas. For users, this aspect enforces live reconfiguration of networks of things, as we already experience in our daily life.

Despite promising results we experienced while using our DSL on small examples with our industrial partners, we acknowledge that many challenges remain. First, reconciling high-level device capabilities with low-level complex communication frameworks available for the plethora of devices will require Complex Event Processing, a now mature field with powerful techniques. Second, evaluating our declarative sublanguages, for network configurations and business rules, on large-scale deployments will provide us insight on how to improve each sublanguage and identify which patterns need to be integrated into libraries to facilitate such definitions. Finally, non-functional properties need to be enforced through appropriate code generation both in a centralised and distributed configurations.

REFERENCES

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376.

Bhandari, S. R. and Bergmann, N. W. (2013). An internet-of-things system architecture based on services and events. In *2013 IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pages 339–344.

Bousse, E., Degueule, T., Vojtisek, D., Mayerhofer, T., Deantoni, J., and Combemale, B. (2016). Execution framework of the gemoc studio (tool demo). In *Proceedings of the 2016 ACM SIGPLAN International Conference on Software Language Engineering*, SLE 2016, pages 84–89, New York, NY, USA. ACM.

Brandtzæg, E., Mohagheghi, P., and Mosser, S. (2012). Towards a domain-specific language to deploy applications in the clouds. In *Third International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 213–218.

Chaqfeh, M. A. and Mohamed, N. (2012). Challenges in middleware solutions for the internet of things. In *2012 International Conference on Collaboration Technologies and Systems (CTS)*, pages 21–26.

Cheng, B., Zhu, D., Zhao, S., and Chen, J. (2016). Situation-aware iot service coordination using the event-driven soa paradigm. *IEEE Transactions on Network and Service Management*, 13(2):349–361.

Cugola, G. and Margara, A. (2012). Complex event processing with t-rex. *Journal of Systems and Software*, 85(8):1709–1728.

García, C. G., G-Bustelo, B. C. P., Espada, J. P., and Cueva-Fernandez, G. (2014). Midgar: Generation of heterogeneous objects interconnecting applications. a domain specific language proposal for internet of things scenarios. *Computer Networks*, 64:143 – 158.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660.

Harrand, N., Fleurey, F., Morin, B., and Husa, K. E. (2016). Thingml: A language and code generation framework for heterogeneous targets. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, MODELS '16, pages 125–135, New York, NY, USA. ACM.

Lee, I. and Lee, K. (2015). The internet of things (iot): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4):431 – 440.

Ma, M., Wang, P., and Chu, C. H. (2013). Data management for internet of things: Challenges, approaches and opportunities. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 1144–1151.

Mosser, S., Collet, P., and Blay-Fornarino, M. (2014). Exploiting the internet of things to teach domain-specific languages and modeling: The arduinoml project. In Demuth, B. and Stikkolorum, D. R., editors, *Proceedings of the MODELS Educators Symposium co-located with the ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014), Valencia, Spain, September 29, 2014.*, volume 1346 of *CEUR Workshop Proceedings*, pages 45–54. CEUR-WS.org.

Mukhopadhyay, S. C., editor (2014). *Internet of Things: Challenges and Opportunities*, volume 9 of *Smart Sensors, Measurement and Instrumentation*, pages 1–17. Springer, Cham.

Munnelly, J. and Clarke, S. (2008). A domain-specific language for ubiquitous healthcare. In *2008 Third International Conference on Pervasive Computing and Applications*, volume 2, pages 757–762.

Pradhan, S. M., Dubey, A., Gokhale, A., and Lehofer, M. (2015). Chariot: A domain specific language for extensible cyber-physical systems. In *Proceedings of the Workshop on Domain-Specific Modeling*, DSM 2015, pages 9–16, New York, NY, USA. ACM.

Salihbegovic, A., Eterovic, T., Kaljic, E., and Ribic, S. (2015). Design of a domain specific language and ide for internet of things applications. In *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 996–1001.

Singh, D., Tripathi, G., and Jara, A. J. (2014). A survey of internet-of-things: Future vision, architecture, challenges and services. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 287–292.

Tan, L. and Wang, N. (2010). Future internet: The internet of things. In *ICACTE*, volume 5.

Xu, L. D., He, W., and Li, S. (2014a). Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243.

Xu, T., Wendt, J. B., and Potkonjak, M. (2014b). Security of iot systems: Design challenges and opportunities. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 417–423.