
7 IoT System Development Methods

Görkem Giray, Bedir Tekinerdogan, and Eray Tüzün

CONTENTS

7.1	Introduction	141
7.2	Background.....	142
7.2.1	System Development Methods	142
7.2.2	IoT System Building Blocks	143
7.3	IoT SDMs in the Literature.....	144
7.3.1	IgniteIoT Methodology	145
7.3.2	IoT Methodology.....	146
7.3.3	IoT Application Development.....	147
7.3.4	ELDAMeth	147
7.3.5	Software Product Line Process to Develop Agents for the IoT	148
7.3.6	General Software Engineering Methodology for IoT	150
7.4	Evaluation of IoT SDMs	151
7.4.1	Method Artifacts.....	152
7.4.2	Process Steps	153
7.4.3	Support for Life Cycle Activities	153
7.4.4	Coverage of IoT System Elements	156
7.4.5	Design Viewpoints.....	156
7.4.6	Stakeholder Concern Coverage	156
7.4.7	Metrics	156
7.4.8	Addressed Discipline	157
7.4.9	Scope	157
7.4.10	Process Paradigm	157
7.4.11	Rigidity of the Method	157
7.4.12	Maturity of the Method	157
7.4.13	Documentation of the Method	158
7.4.14	Tool Support	158
7.5	Conclusion	158
	References.....	159

7.1 INTRODUCTION

It is generally believed that the application of methods plays an important role in developing quality systems. A development method is mainly necessary for structuring the process in producing large-scale and complex systems that involve high costs. Similar to the development of other systems, it is important for IoT systems to be developed in a systematic manner in order to achieve a proper system with respect to both the functional and nonfunctional requirements. Development methods for IoT systems are more complex than traditional software systems and possess challenges from the process perspective. So far, several IoT system development methods (SDMs) have already been proposed in the literature, but an overview and evaluation of SDMs for IoT is still missing.

In this chapter, an overview of SDMs dedicated for IoT systems is presented. For this, the key IoT SDMs (the Ignite | IoT Methodology [Slama et al., 2016], the IoT Methodology [Collins, 2017], IoT Application Development [Patel, 2014], ELDAMeth [Fortino and Russo, 2012], a Software Product Line Process to Develop Agents for the IoT [Ayala and Amor, 2012], and a General Software Engineering Methodology for IoT [Zambonelli, 2016]) found in the literature were identified. Based on these methods, a set of evaluation criteria that include necessary features for characterizing and evaluating IoT SDMs were presented.

The remainder of the chapter is organized as follows. In Section 7.2, a background on SDMs and IoT systems is presented. Section 7.3 summarizes IoT SDMs along with their process flows represented using Business Process Model and Notation (BPMN), and Section 7.4 presents the characterization of these methods using the evaluation criteria. Finally, Section 7.5 concludes the chapter.

7.2 BACKGROUND

System development is generally a complex endeavor, especially for the systems including software, hardware, and communication components. To reduce and control this complexity, many SDMs have been proposed. The use of such methods brings some benefits. IoT system development is also a complex endeavor and encompasses dealing with a diverse set of components. The objective of an IoT SDM is to guide a project team in developing and combining these components in order to be able to fulfill user requirements.

7.2.1 SYSTEM DEVELOPMENT METHODS

An SDM is an approach to develop a system systematically based on directions and rules (Brinkkemper, 1996). In this chapter, SDM refers to the development of an IoT system. SDM is preferred over the “software development method” concept since an IoT system encompasses many software, hardware, and communication components. Therefore, the development of an IoT system calls for more holistic methods than software development endeavors. Since IoT systems have software components as well, SDMs can also contain and/or benefit from software development methods.

It is generally believed that the application of SDMs plays an important role in developing quality systems. SDMs provide many benefits, including

- An SDM provides engineers with a set of guidelines for developing some artifacts and their verification against the requirements defined in a problem statement.
- Since SDMs formalize certain procedures of design and externalize design thinking, they help to avoid the occurrence of overlooked issues in the development process and tend to widen the search for appropriate solutions by encouraging and enabling the engineer to think beyond the first solution that comes to mind.
- SDMs help to provide logical consistency among the different processes and phases in the development process. This is particularly important for the development of large and complex systems, which are produced by large teams of designers and developers. A development method provides a set of common standards, criteria, and goals for the team members.
- An SDM helps reduce possible errors in the development process and provides heuristic rules for evaluating design decisions.
- Mainly from the organizational point of view, an SDM helps to identify important progress milestones. This information is necessary to control and coordinate the different phases in system development.

An SDM is mainly necessary for structuring the process in producing large-scale and complex systems that involve high costs. The motivation for SDMs can thus be summarized as

directing the project team, leading to better systems by considering various solution alternatives, providing consistency among different processes, reducing failures, and identifying important milestones.

7.2.2 IoT SYSTEM BUILDING BLOCKS

An IoT system is made up of various components interacting with each other. Figure 7.1 provides a conceptual model of the components making up a typical IoT system, including the relations among the basic IoT concepts. The model has been adopted from the Alliance for Internet of Things Innovation (AIOTI) domain model (AIOTI, 2016). The domain model represents the basic concepts and relationships in the domain at the highest level. Since the AIOTI domain model includes intuitive high-level basic concepts, it has been also referenced by some other standardization efforts, such as the IEEE P2413 Working Group, whose objective is to form some standards to support the IoT’s growth.

In the model, a user interacts with a physical entity from the physical world, that is, a thing. The user can be a human person or a software agent that has a goal. To achieve this goal, an interaction with the physical environment is needed, and this interaction is performed through the mediation of an IoT system. A thing is a discrete, identifiable part of the physical environment that can be of interest to the user for the completion of his goal. Things can be any physical entities, such as humans, cars, animals, or computers.

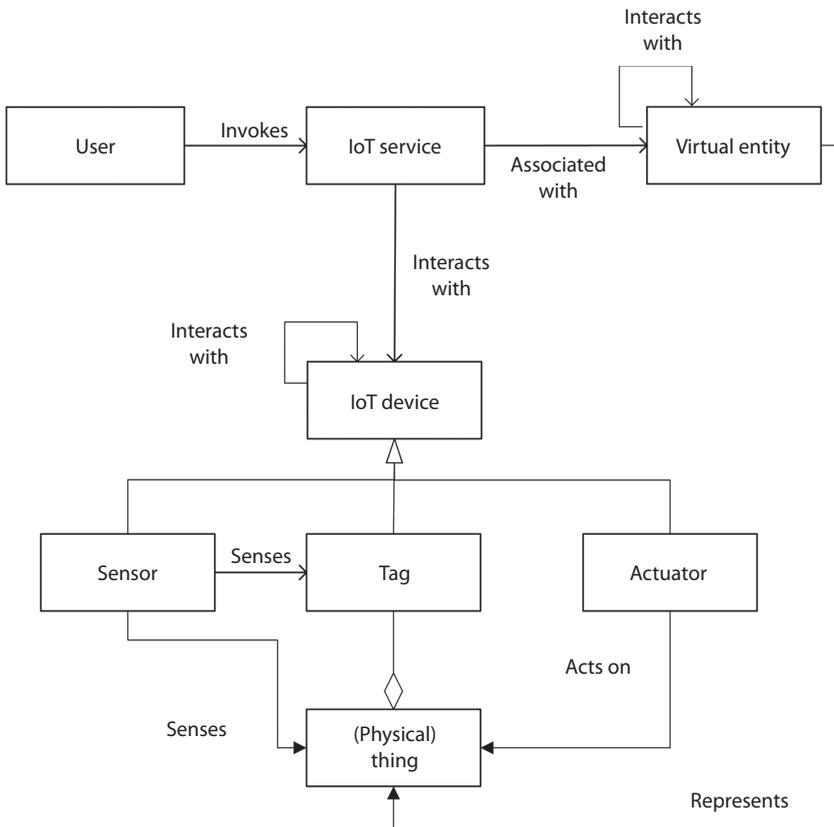


FIGURE 7.1 Conceptual model of a typical IoT system. (Adapted from Alliance for Internet of Things Innovation, High Level Architecture [HLA], Release 2.1, AIOTI WG03—IoT Standardisation, 2016.)

The interaction between a user and a thing in the IoT is mediated by a service, which constitutes the interface between a user and an IoT system. A service is associated with a virtual entity, that is, a digital representation of the physical entity. A thing can be represented in the digital world by a virtual entity. Different kinds of digital representations of things can be used, such as objects, three-dimensional models, avatars, or even social network accounts. Some virtual entities can also interact with other virtual entities to fulfill their goal.

An important aspect in IoT is that changes in the properties of a thing and its corresponding virtual entity need to be synchronized. This is usually realized by a device that is embedding into, attached to, or simply placed in close vicinity of the thing. In principle, three types of devices can be identified, namely, sensors, tags, and actuators. Sensors are used to measure the state of things they monitor. Essentially, sensors take a mechanical, optical, magnetic, or thermal signal and convert this into voltage and current. This provided data can then be processed and used to define the required action. Tags are devices used to support the identification process, typically using specialized sensors called readers. The identification process can take place in different forms, including optical, as in the case of barcodes and QR code, or radio frequency based. Actuators, on the other hand, are employed to change or affect the things.

7.3 IoT SDMS IN THE LITERATURE

To identify the different IoT SDMs, a thorough domain analysis process has been applied, in which relevant studies were searched in the literature that directly propose an IoT SDM or partially address the development of one or more components of an IoT system. As a result, six SDMs were identified. In the domain analysis, two basic activities can be distinguished, namely, domain scoping and domain modeling. In the scoping process, the scope of the domain analysis process is defined and the set of knowledge sources is selected. In this text, the scope consists of the identified IoT SDMs, as listed in Table 7.1. In the domain modeling process, the SDMs are briefly introduced, along with their process flows, illustrated using BPMN, and these SDMs are evaluated against some criteria.

TABLE 7.1
IoT SDMs in Literature

Method	Abbreviation	Origin	Base
The Ignite IoT Methodology (Slama et al., 2016)	Ignite	Industry	Best practices from the projects in the industry; project management guidelines, such as Project Management Body of Knowledge (PMBOK)
The IoT Methodology (Collins, 2017)	IoT-Meth	Industry	Best practices from the projects in the industry
IoT Application Development (Patel and Cassou, 2015; Patel, 2014)	IoT-AD	Academia	Built on macroprogramming approach; inspired by model-driven design
ELDAMeth (Fortino and Russo 2012; Fortino et al., 2014, 2015)	ELDAMeth	Academia	Multiagent system development
A Software Product Line (SPL) Process to Develop Agents for the IoT (Ayala and Amor 2012; Ayala et al., 2012, 2014, 2015)	SPLP-IoT	Academia	Multiagent system development; SPLE
A General Software Engineering (SE) Methodology for IoT (Zambonelli, 2016, 2017)	GSEM-IoT	Academia	Traditional software development; abstractions from IoT systems

Note: SPLE, Software Product Line Engineering.

7.3.1 IGNITE|IoT METHODOLOGY

Derived from real-world IoT projects, the Ignite | IoT Methodology (Slama et al., 2016) (“Ignite” in this chapter) is aimed at various IoT stakeholders, including product managers, project managers, and solution architects. The methodology has two major groups of activities: IoT Strategy Execution and IoT Solution Delivery. IoT Strategy Execution aims to define an IoT strategy and a project portfolio supporting this strategy. IoT Solution Delivery supports IoT solution design and IoT project management, along with some artifacts, such as project templates, checklists, and solution architecture blueprints. These two groups of activities should be synchronized to keep the project portfolio in line with the strategy and revise the strategy according to the outcomes of the project portfolio. Figure 7.2 illustrates the process flow of Ignite.

IoT Strategy Execution is about business perspective and involves identifying an opportunity, developing a business model, and making decisions on how to manage these opportunities (such as internal project, external acquisition, and spin-off).

IoT Solution Delivery is about delivering a solution, which is conceptually defined during the IoT Strategy Execution phase, and has a life cycle consisting of the planning, building, and running of the system. Planning starts with project initiation, in which an initial solution design and a project organization chart are delivered. Moreover, stakeholder, environment, requirements, risk, and resource analysis should be conducted. After the initiation, the tasks are managed under seven work streams: (1) project management, (2) crosscutting tasks, (3) solution infrastructure and operations, (4) back-end services, (5) communication services, (6) on-asset components, and (7) asset preparation. Project management encompasses the activities for initiating, planning, executing, monitoring, controlling, and closing a project. Crosscutting tasks address the dependencies among subsequent work streams, such as security and testing. Solution infrastructure and operations include the

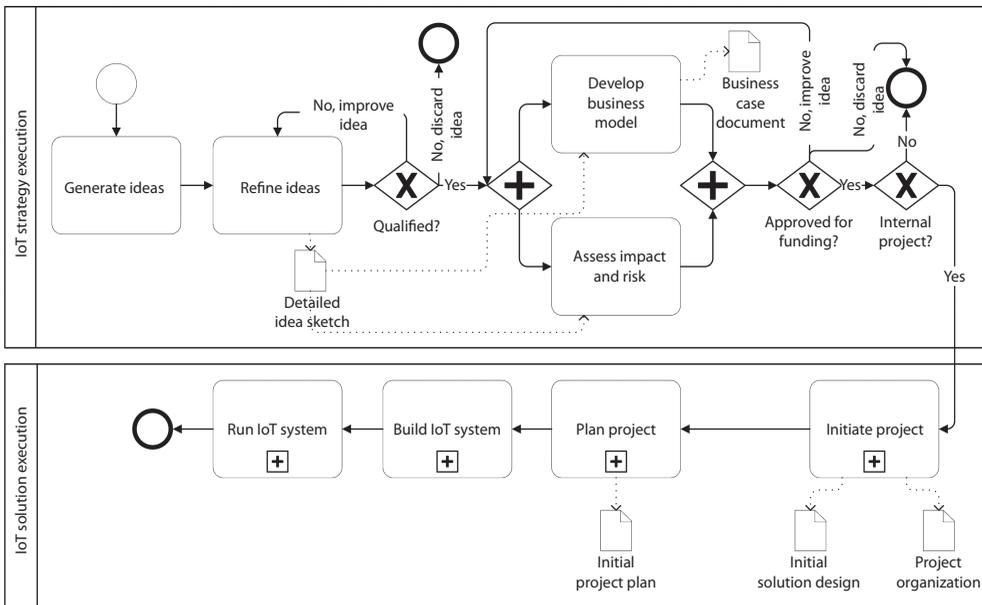


FIGURE 7.2 Ignite process flow.*

* The process flows are illustrated using BPMN, which provides a standard graphical notation for representing processes. For more information, see <http://www.bpmn.org/>.

- : Exclusive gateway, which means that only one of the following branches can be traversed.
- : Parallel gateway, which means that the following tasks can be performed in parallel.

installation and management of hardware and software infrastructure, on which an IoT system will be developed and operated. Back-end services refer to the IoT services typically hosted on a private or public cloud and interacting with IoT devices. Communication services encompass the installation and management of communication infrastructure. On-asset components refer to the development or procurement of software and the manufacturing or procurement of hardware and network components to be integrated with a thing in an IoT system. Asset preparation addresses the manufacturing and procurement of a thing in an IoT system.

7.3.2 IoT METHODOLOGY

The IoT Methodology (Collins, 2017) (“IoT-Meth” in this chapter) is a generic, lightweight method built on iterative prototyping and Lean start-up approaches. Figure 7.3 illustrates the steps of IoT-Meth, eliminating its iterative nature for the sake of simplicity (the steps are renamed for better understanding). IoT-Meth involves the following iterative steps (the original names of the steps are in parenthesis):

1. *Generate ideas (cocreate)*: This step involves the identification of problem areas by communicating with stakeholders, especially end users. The objective is to generate ideas on potential problems from a business perspective. Some of these ideas are elaborated with their use cases, to be refined in the next step.
2. *Refine ideas (ideate)*: Some of the ideas identified in the former step are further elaborated to be communicated with project managers, designers, and implementers. The artifact named IoT Canvas can be used in brainstorming sessions with stakeholders to identify and validate high-level requirements. IoT Canvas mainly consists of a problem statement, key actors, things in the physical environment, sensors and actuators, data models, middleware

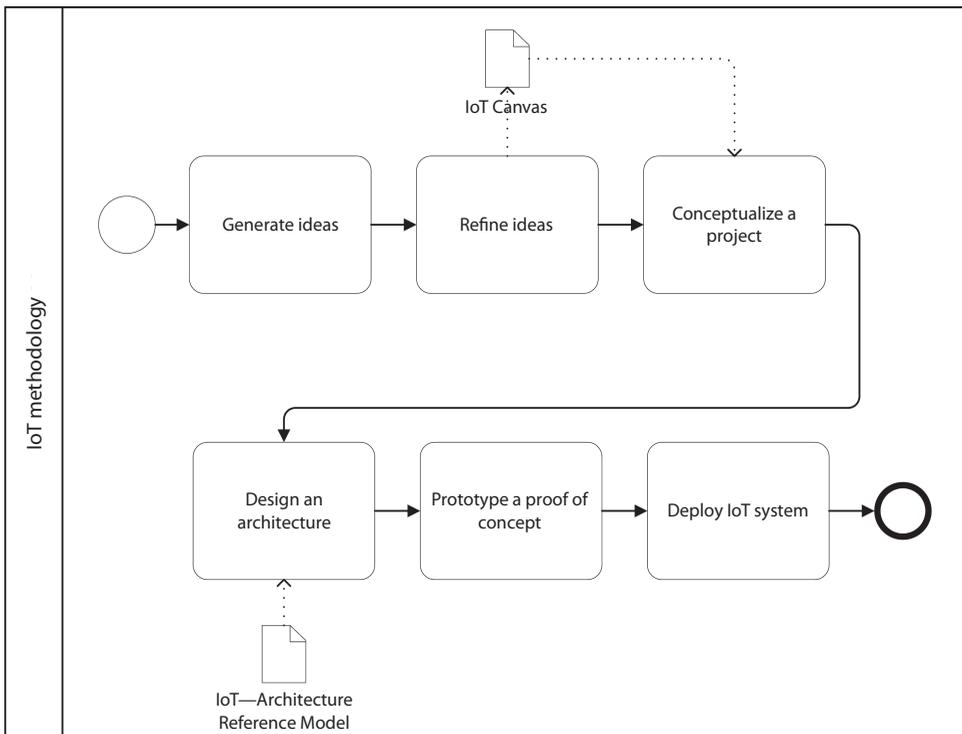


FIGURE 7.3 IoT-Meth process flow.

requirements to connect IoT services, third-party web services to be integrated, and user interface (UI) sketches.

3. *Conceptualize a project (Q&A)*: This step involves analyzing refined ideas further to close the gap between idea and implementation. The requirements are analyzed and validated; the domain is analyzed further.
4. *Design an architecture (IoT OSI)*: In this step, the requirements are mapped to an architecture and infrastructure. The artifact named IoT—Architecture Reference Model is an input for this step and basically an adaptation of the seven-layer International Organization for Standardization/Open Systems Interconnection (ISO/OSI) reference model for IoT solutions. This reference model comprises five layers: end points, connectivity, middleware, IoT services, and applications. These layers help in classifying components and hence managing complexity.
5. *Prototype a proof of concept (prototype)*: This step encompasses building prototypes and iterating toward minimal viable IoT systems. The prototypes are assessed, and forthcoming iteration plans are revised accordingly.
6. *Deploy IoT system (deploy)*: The last step is about deploying the system and closing the feedback loop. Generally, the system is improved continuously according to the feedback.

IoT-Meth does not define well-defined roles with descriptions and responsibilities. It only addresses some roles, such as end user, designer, implementer, and project manager, without any detail.

7.3.3 IoT APPLICATION DEVELOPMENT

Patel and Cassou (2015; Patel, 2014) propose an approach to IoT application development (“IoT-AD” in this chapter) built on macroprogramming (in contrast to node-centric programming), in which the behavior of a system is specified using high-level abstractions and then compiled to node-level code (Pathak et al., 2007). IoT-AD encompasses a development methodology and a concrete development framework realizing this methodology. IoT-AD is built on the separation of concerns principle and classifies the IoT domain into four areas of concern: (1) domain, (2) functional, (3) deployment, and (4) platform. The domain concern is related to domain-specific concepts of an IoT system and mainly encompasses the identification of a domain vocabulary. The functional concern is about specifying an architecture and implementation of this architecture. The deployment concern encapsulates deployment specifications of each thing. The platform concern addresses the development of platform-specific device drivers for each type of thing. The process flow of IoT-AD organized according to the concerns is illustrated in Figure 7.4. IoT-AD provides a set of modeling languages for modeling the concepts in these areas of concern and some automation techniques for reducing development effort.

7.3.4 ELDAMETH

ELDAMeth (Event-driven Lightweight Distilled state charts-based Agents Methodology) utilizes an agent-based paradigm to guide the development of software for things in an IoT system (Fortino and Russo, 2012). Agents can be defined as autonomous software components acting in a distributed, networked environment. Therefore, agents are very similar to the software components of things operating in a distributed, networked environment within an IoT system. Things are named smart objects (SOs), referring to the autonomy, which is an important feature of agents. SOs (refers to the software component of things) are treated as the fundamental building blocks of IoT systems (Fortino et al., 2015).

As illustrated in Figure 7.5, this methodology mainly encompasses three main phases (Fortino and Russo, 2012; Fortino et al., 2014): (1) The modeling phase takes high-level design models and requirements as inputs and produces a detailed design of SOs, which can be translated into

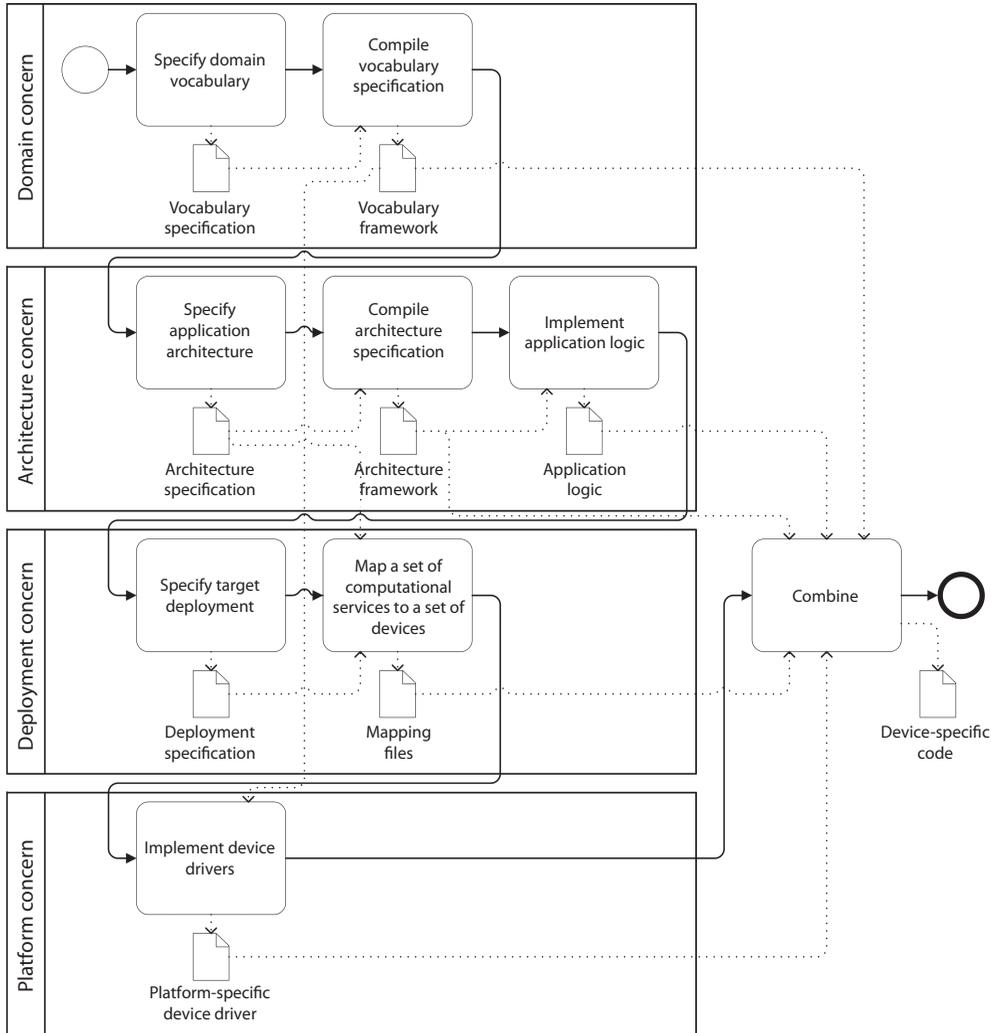


FIGURE 7.4 IoT-AD process flow.

platform-independent code. (2) The simulation phase takes requirements and platform-independent ELDA SO code as inputs and produces simulation results to be evaluated against requirements. (3) The implementation phase encompasses the development and test of platform-specific ELDA SO code.

ELDAMeth focuses on the technical aspects of developing SOs. The business aspects, as well as requirements engineering for developing IoT systems, are not addressed in this methodology.

7.3.5 SOFTWARE PRODUCT LINE PROCESS TO DEVELOP AGENTS FOR THE IoT

The Software Product Line Engineering (SPLE) paradigm aims to develop software by identifying commonalities and variabilities of a family of software products. Commonalities form a reusable platform, and variabilities refer to the specific features of particular software within the scope of a family of software products. SPLE separates two main processes (Pohl et al., 2005): Domain engineering is for establishing the platform with common features, and application engineering is for deriving particular applications built on top of the platform. There are several motivations for SPLE reported in the literature (Pohl et al., 2005), such as reduction of development costs, enhancement of

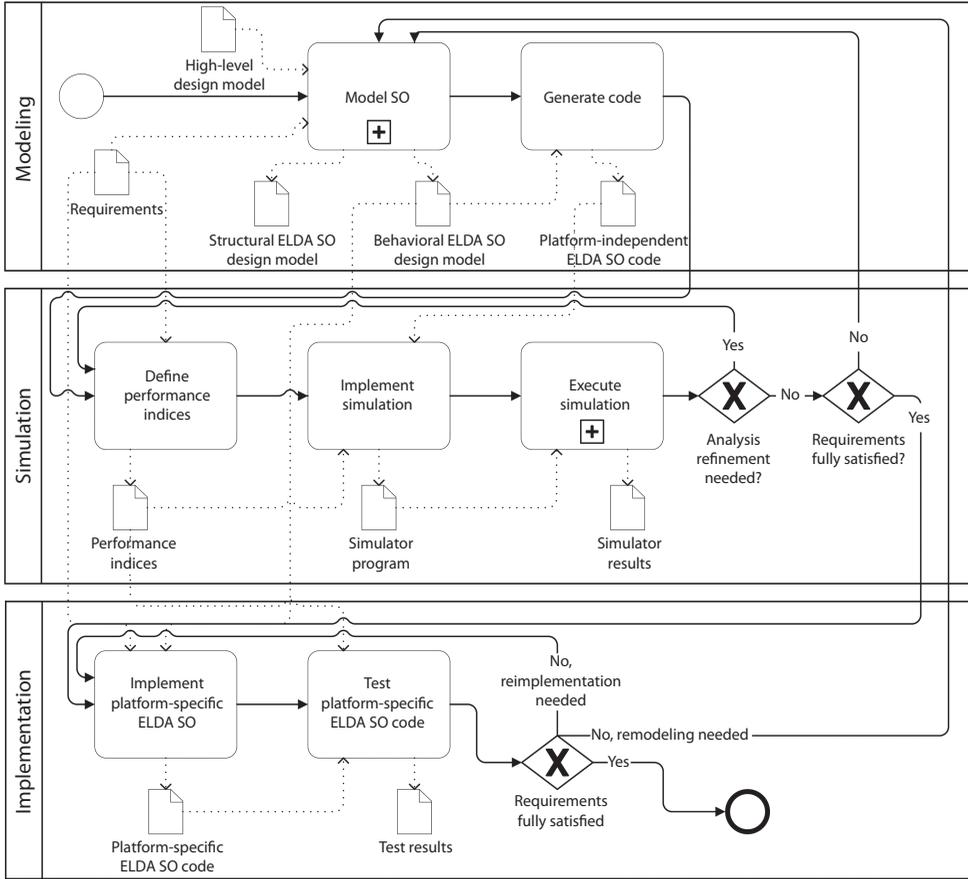


FIGURE 7.5 ELDAMeth process flow.

quality, reduction of time to market, reduction of maintenance effort, coping with evolution, coping with complexity, and improving cost estimation.

IoT systems envision a large-scale, complex, heterogeneous network of things. Therefore, agents, which are autonomous software components, can be an enabler for self-managed IoT systems (Ayala et al., 2015). Developing software components of things using an agent-based paradigm is considered due to the agents’ distributed nature, context awareness, and self-adaptation (Ayala et al., 2015).

A Software Product Line Process to Develop Agents for the IoT (“SPLP-IoT” in this chapter) combines SPLE with an agent-based paradigm to utilize the benefits of both of these paradigms for developing IoT systems (Ayala et al., 2015). The aim is to identify commonalities among software agents and develop common reference architecture, and hence obtain a reduction in implementation time and cost, as well as an increase in quality.

Figure 7.6 illustrates an overview of SPLP-IoT. Domain engineering is responsible for establishing a reusable platform and thus defining commonalities and variabilities of a multiagent system. This is achieved by mining domain requirements, analyzing the IoT multiagent system domain, and specifying and realizing domain variability. Domain variability defines the variation points to be configured to obtain specific agents. Domain variability is specified by an IoT multiagent system variability model represented using Common Variability Language (CVL). CVL is a domain-independent language defined using a Meta-Object Facility (MOF)–based metamodel, which is used for specifying and resolving variability (Haugen et al., 2013). An IoT multiagent system architecture, which represents commonalities and variabilities, is produced after the variability model is defined.

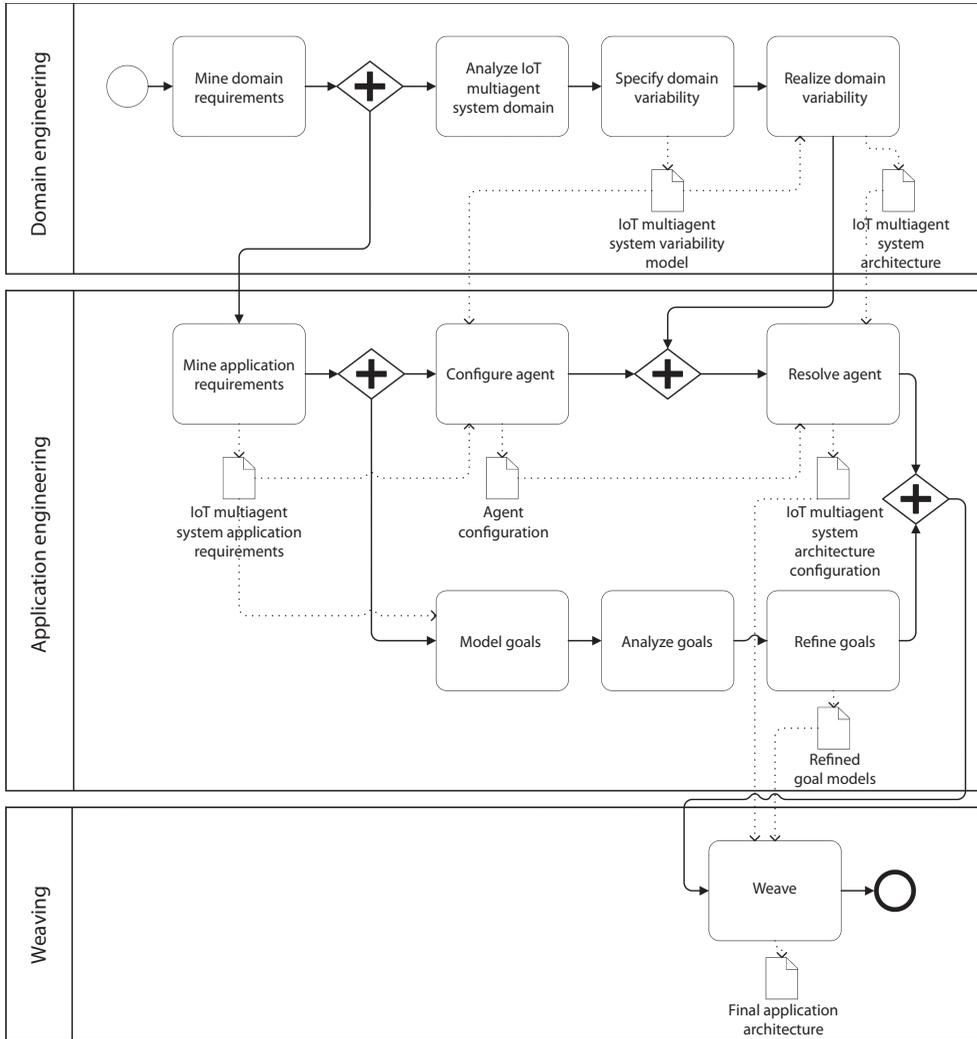


FIGURE 7.6 SPLP-IoT process flow.

Therefore, an IoT multiagent system architecture acts as a base model, forming both the common platform and the variability points to be configured for each particular agent. Application engineering encompasses building agents by exploiting the variability model and IoT multiagent system architecture. In this part, an agent is configured according to specific application requirements and an agent configuration is obtained as a result. When resolving the agent, an IoT multiagent system architecture configuration is generated by a CVL tool. Modeling and analyzing goals are for checking consistency among agent goals, context, and plans. Afterwards, the goals are refined. Refined goal models and IoT multiagent system architecture configuration are processed to produce a final application architecture. This final application architecture corresponds to a software component on a thing, and this software component is built on top of a platform (built around commonalities).

7.3.6 GENERAL SOFTWARE ENGINEERING METHODOLOGY FOR IoT

Zambonelli (2016, 2017) proposes some general guidelines and steps of a general software engineering methodology for developing IoT systems (“GSEM-IoT” in this chapter). GSEM-IoT consists

of three phases (Zambonelli, 2016): (1) analysis phase, which includes the identification and analysis of actors, existing infrastructure, functionality, and requirements; (2) design phase, which includes the design of avatars, groups, and coalitions and identification of new infrastructural needs; (3) and implementation phase, which includes the implementation of avatars and coordinators, along with the deployment of new “things” and new middleware. Figure 7.7 illustrates the overview of GSEM-IoT, which is attributed as a first small step toward a general method for developing IoT systems.

7.4 EVALUATION OF IoT SDMS

An SDM guides a system development endeavor by providing guidance on many different aspects, such as addressing stakeholder concerns, steps to be followed, and artifacts to be produced. Figure 7.8 illustrates a conceptual model of the important concepts related to an SDM. This model is adapted to an IoT context. In essence, each IoT system will be important for a number of stakeholders that have a number of concerns. From a simplified perspective, an SDM consists of a set of life cycle phases, a number of steps, and artifacts that reflect IoT elements and design viewpoints for modeling various design perspectives. Further, an SDM is focused on a particular paradigm and is supported by documentation and tools.

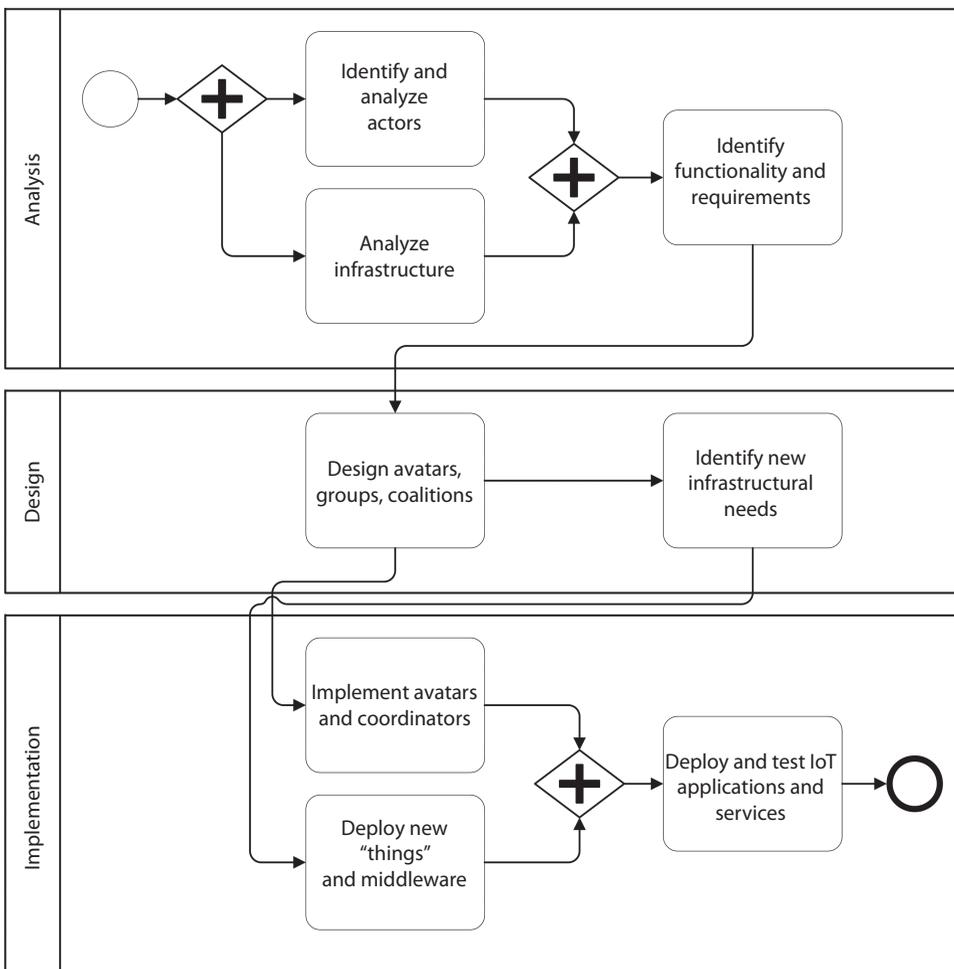


FIGURE 7.7 GSEM-IoT process flow.

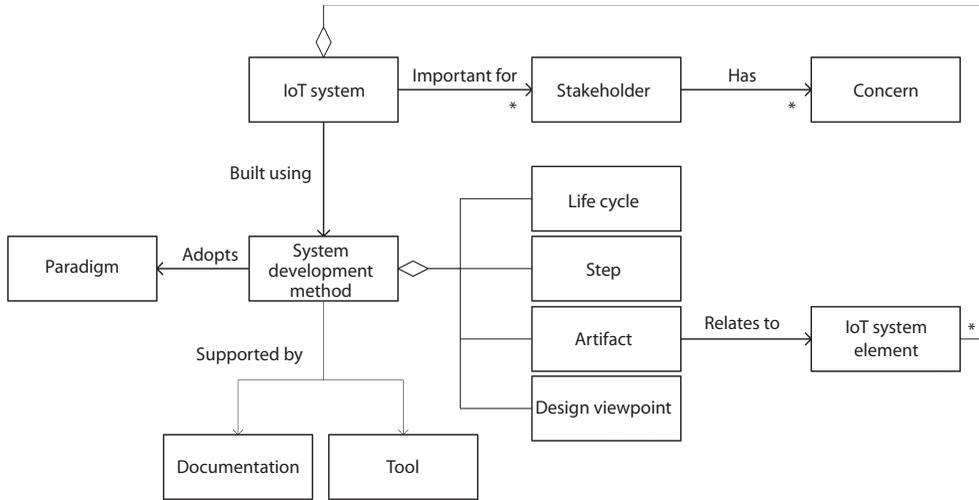


FIGURE 7.8 Conceptual model for a method and related concepts.

An SDM can be evaluated against the concepts illustrated in Figure 7.8. Table 7.2 shows the evaluation criteria to analyze existing IoT SDMs based on the conceptual model of Figure 7.8. The evaluation criteria relate to the support for SDM elements, as well as more qualitative issues with respect to the use and pragmatics of the SDM.

7.4.1 METHOD ARTIFACTS

Table 7.3 lists the artifacts defined in the documentations of the SDMs. Ignite provides many artifacts addressing different aspects of an IoT system development project. Most of the artifacts deal with the business and project management aspects. Most of the artifacts are well known from the software engineering discipline, such as business case document and domain model. Ignite also gives some examples of specific formats to be used in some phases. For instance, Innovation Project Canvas* can be used in the idea refinement phase, whose output is a detailed idea sketch. Innovation Project Canvas provides a format (project title and objectives, customers, customer needs, market trends, competition, value proposition and product description, solution, business model, challenges and risks, critical unknowns, key activities, and issues to be treated first) for specifying ideas, and hence provides a predefined format for producing a detailed idea sketch artifact. IoT-Meth includes mainly two artifacts: (1) IoT Canvas, which is a template to be used in brainstorming sessions to identify and validate feasible ideas, and (2) IoT—Architecture Reference Model, against which a solution’s architecture is mapped. IoT-AD proposes to use different kinds of artifacts addressing different viewpoints of an IoT application, for instance, the vocabulary specification artifact for the domain viewpoint; the architecture specification, architecture framework, and application logic artifacts for the architecture viewpoint; the deployment specification and mapping file artifacts for the deployment viewpoint; and the vocabulary framework artifact for the platform viewpoint. ELDAMeth provides artifacts to develop software for IoT devices (SOs in ELDAMeth terminology), including some artifacts for simulation. The artifacts of both IoT-AD and ELDAMeth only cover software development for IoT devices; they do not address the development of IoT services or the hardware and communication components. SPLP-IoT uses some artifacts from SPLE research addressing commonalities and variabilities. For instance, the

* Innovation Project Canvas website, www.innovationprojectcanvas.com.

TABLE 7.2
Evaluation Criteria for Analyzing IoT SDMs

Criteria	Description
Method artifacts	What are the method artifacts in the overall process?
Process steps	What are the process steps?
Support for life cycle activities	Which life cycle activities are supported by the method? (feasibility analysis, requirements definition, design, development, testing, deployment, project management, maintenance)
Coverage of IoT system elements	Is the process related to all the IoT system elements? (e.g., sensors and actuators)
Design viewpoints	Does the method include different design viewpoints?
Stakeholder concern coverage	Does the method support the required stakeholder concerns (expectations from the IoT system, expectations from the project realizing the IoT system)?
Metrics	Does the method provide any metrics? (such as requirements quality metrics, code quality metrics, project management metrics, test metrics, etc.)
Addressed discipline	What is the addressed engineering discipline? (system, software, mechanical, etc.)
Scope	What is the scope of the method? General purpose vs. domain specific?
Process paradigm	What is the adopted process paradigm? (plan-driven vs. agile)
Rigidity of the method	Is the method extensible? (process, rules, artifacts)
Maturity of the method	Has the method been validated?
Documentation of the method	How well is the method documented?
Tool support	Does the method have tool support? If so, which?

IoT multiagent system variability model represents the possible variability points for each agent (software component of IoT device in this case). Moreover, the IoT multiagent system architecture includes a blueprint of a common platform (besides variability points) to establish a software product line. No artifact has been defined for GSEM-IoT yet, as the method is still under development (Zambonelli, 2016).

7.4.2 PROCESS STEPS

Both Ignite and IoT-Meth define a high-level process flow addressing the development of an IoT system from idea to a running system (Figures 7.2 and 7.3). IoT-AD, ELDAMeth, and SPLP-IoT include a process flow for developing software to be deployed on IoT devices (Figures 7.4 through 7.6). ELDAMeth has some steps for simulation, which aims to validate the solution design before implementation. SPLP-IoT addresses commonality and variability analysis for SPL with some steps. GSEM-IoT provides a process flow for systematic development of IoT systems (Figure 7.7). GSEM-IoT tries to enrich the traditional software development process with key concepts and abstractions from the IoT domain.

7.4.3 SUPPORT FOR LIFE CYCLE ACTIVITIES

Ignite addresses the whole life cycle of developing an IoT system, focusing more on project management, feasibility analysis, requirements engineering, and analysis and design. Ignite also includes software development and testing activities; on the other hand, it does not provide any technical details on how to develop and test software for IoT systems. IoT-Meth partially addresses feasibility analysis, requirements engineering, analysis and design, and deployment. IoT-Meth does not provide any details on these life cycle activities. IoT-AD, ELDAMeth, SPLP-IoT, and GSEM-IoT focus on requirements engineering, analysis and design, and development at varying levels of detail. In addition, IoT-AD

TABLE 7.3
Artifacts Defined in the Identified SDMs

SDM	Artifact	Description
Ignite	Detailed idea sketch	Describes the key elements of an idea to be elaborated for further evaluation
	Business case document	Refinement and validation of a detailed idea sketch; evaluated for funding
	Project organization	Defines the teams involved and how these teams are structured to achieve the objective of a project
	Initial project plan	Formed after funding decision, based on the idea and requirements
	Problem statement	Contains a short description of the problem domain and vision for the IoT solution
	Stakeholder analysis	Artifact including an analysis of the stakeholders by their interest and influence on the project
	Site survey	A document covering all aspects of an IoT device (hardware) to be manufactured or procured
	Solution sketch	Narrows down the solution scope and creates a basis for the communication between business and technical stakeholders
	Project dimensions	Capture all important aspects of an IoT solution and made up of five main dimensions: (1) assets and devices, (2) communications and connectivity, (3) back-end services, (4) standards and regulatory compliance, and (5) project environment
	Quantity structure	Includes projections for possible changes on the numbers of users, assets, etc.
	Milestone plan	Defines the key milestones of the project
	Process maps/use cases	Demonstrate how an IoT solution addresses customer's problem
	UI mock-ups	Visualizations of key UIs; provide a basis for discussing ideas and validating requirements with end users and business stakeholders
	Domain model	Encompasses a business-oriented, consolidated view of the key data entities of an IoT solution
	Asset integration architecture	Describes the relationships between assets and devices (things) and the back-end
	SOA landscape	Describes the key software components and their main business functions; different than asset integration architecture in being technology agnostic and focusing on business functions
	Software architecture	Defines the key software components and their relationships
Technical infrastructure	High-level view of an IoT solution and its environment; contains assets, communication infrastructure, etc.	
Hardware design	Addresses the main components of an asset on an IoT device, such as CPU, memory, power supply, digital I/O, and communication modules	
IoT-Meth	IoT Canvas	Defines high-level characteristics of an IoT solution for providing a basis for further assessment (including funding); can include a problem statement, things in the physical environment, key actors, etc.
	IoT—Architecture Reference Model	Defines the layers identifying an important aspect of an IoT solution; such layers can include connectivity, middleware, IoT services, and applications
IoT-AD	Vocabulary specification	Defines resources, which are conceptual representations of sensors, actuators, storages, and UIs
	Vocabulary framework	Contains concrete classes and interfaces corresponding to resources defined in a vocabulary specification
	Architecture specification	Guides software developers in developing software components of IoT devices
Architecture framework	Contains abstract classes that hide interaction details with other software components and allow software developers to focus on the application logic of a particular software component	

(Continued)

TABLE 7.3 (CONTINUED)
Artifacts Defined in the Identified SDMs

SDM	Artifact	Description
	Application logic	Represents a concrete implementation of abstract classes defined in an architectural framework in line with the architecture specification
	Deployment specification	Includes the details of each IoT device, such as resources (sensor, actuator, storage, and UI) hosted by devices and types of devices
	Mapping files	Include the mapping information between software components and IoT devices
	Platform-specific device driver	Concrete implementation of some functionality specific to an IoT device, such as reading a barcode using an IoT device with an Android operating system
ELDAMeth	Device-specific code	Executed on each particular IoT device
	Requirements	Define functional and nonfunctional requirements of a multiagent system under development
	High-level design models	Represent a solution without any details based on requirements
	Structural ELDA SO design models	Class diagrams representing the interaction relationships among agents and roles
	Behavioral ELDA SO design models	State charts representing agent behaviors and/or roles
	Platform-independent ELDA SO code	Part of the code that is independent of any specific technology
	Platform-specific ELDA SO code	Part of the code that is dependent on a specific technology
	Performance indices	Define the criteria against which the results of a simulation will be evaluated
	Simulator program	Enables execution of the simulation
	Simulation results	Findings obtained from execution of the simulation
	Test results	Document the findings obtained from tests, also considering the performance indices
SPLP-IoT	IoT multiagent system variability model	Defines variation points, which can be configured differently when implementing a particular agent
	IoT multiagent system architecture	Defines the main components of a system, along with their relationships
	IoT multiagent system application requirements	Refer to the specific expectations from a particular software component running on an IoT device
	Agent configuration	Refers to the implementation of a particular IoT device by adjusting variation points defined in a variability model
	IoT multiagent system architecture configuration	Contains the set of components and connections that leads to the realization of the final application architecture
	Refined goal models	Contain a set of goals and plans and the context of the agent that are consistent and whose conflicts are detected; these goals define how these agents (IoT devices in this case) will behave
	Final application architecture	Derived from IoT multiagent system architecture configuration and refined goal models
GSEM-IoT	No artifact defined	

covers deployment by proposing to produce a deployment specification, which includes the details of each IoT device. Moreover, IoT-AD claims that its approach supports the maintenance phase. The rationale behind this is the separation of different concerns (domain, architecture, deployment, and platform) and automation techniques provided in the method. ELDAMeth includes a testing activity in the implementation phase and proposes to produce an artifact, including test results.

7.4.4 COVERAGE OF IoT SYSTEM ELEMENTS

Ignite provides guidance for developing IoT services and software components for IoT devices from a business and project management perspective; however, it does not provide a low-level, technical guidance. Ignite excludes the manufacturing of assets (things). IoT-Meth addresses IoT services and software components for IoT devices at an architectural level; however, it does not include any detailed subprocesses for developing each of these elements. IoT-AD, ELDAMeth, and SPLP-IoT only cover the development of software components, which run on IoT devices (things). GSEM-IoT mainly provides a conceptual view (without providing any technical details) on the development of software components of IoT devices.

7.4.5 DESIGN VIEWPOINTS

A design viewpoint examines a system from a particular perspective. A design viewpoint of a system can be represented by one or more artifacts. These artifacts should address the concerns of that particular viewpoint. For instance, a functional viewpoint of a system can focus on the functionalities to be provided by a system, whereas a deployment viewpoint can treat the concerns on the deployment of a system.

Ignite addresses business, usage, functional, and implementation viewpoints. It defines certain artifacts to address these viewpoints:

1. Business: Business case document, site survey, project dimensions, milestone plan
2. Usage: Process maps and use cases
3. Functional: UI mock-ups and domain model
4. Implementation: Software architecture and technical infrastructure

IoT-AD specifies four viewpoints concerning domain, architecture, deployment, and platform (Patel, 2014). The domain viewpoint addresses the specification of a domain-specific vocabulary for an IoT application. The architecture viewpoint encompasses specifying application architecture, compiling architecture specification, and implementing application logic. The deployment viewpoint is about describing deployment specifications for devices and mapping a set of computational services to a set of devices. The platform viewpoint is for implementing platform-specific device drivers. IoT-Meth, ELDAMeth, SPLP-IoT, and GSEM-IoT do not explicitly address different viewpoints.

7.4.6 STAKEHOLDER CONCERN COVERAGE

Ignite proposes to identify stakeholders and analyze their expectations when initiating a project to address their concerns. IoT-Meth discusses communication with stakeholders during the idea generation step, with no further detail. GSEM-IoT has a step for identifying and analyzing actors. IoT-AD, ELDAMeth, and SPLP-IoT do not address stakeholders explicitly.

7.4.7 METRICS

None of the methods offer a metric set to be used during IoT system development for tracking different aspects of the process and product. Ignite states that IoT-specific metrics should be created in an

organization if that organization considers IoT systems as a strategic component (Slama et al., 2016). IoT-AD has been evaluated using some well-known metrics measuring development effort, success of reusability, and code quality (Patel, 2014). On the other hand, IoT-AD does not contain any specific metric to be used when applying the method to a specific project. IoT-Meth, ELDAMeth, SPLP-IoT, and GSEM-IoT do not include any information on metrics.

7.4.8 ADDRESSED DISCIPLINE

Ignite provides a comprehensive view by handling IoT devices and IoT services, ending up with a system engineering perspective to an IoT system development project. Although IoT-Meth mentions IoT system elements, it has a very superficial system engineering perspective. IoT-AD, ELDAMeth, SPLP-IoT, and GSEM-IoT mainly focus on the software engineering aspect of IoT systems. GSEM-IoT partially addresses back-end services from a software engineering perspective as well.

7.4.9 SCOPE

All the methods are designed for general-purposes usage and are not specialized for a specific domain (e.g., agriculture or transportation).

7.4.10 PROCESS PARADIGM

Ignite, according to its documentation (Slama et al., 2016), is compatible with both plan-driven and agile paradigms. The creators of Ignite claim that an agile paradigm can be applied in IoT system development, but on the other hand, they emphasize some issues, which are potentially challenging: (1) scaling agile to large, distributed project organizations; (2) potential cultural differences between hardware and software engineers; (3) long-term planning needed for hardware (IoT device) design, implementation, and testing; and (4) challenging release management due to distributed nature of IoT devices. IoT-Meth does not specifically define a process paradigm, but on the other hand, the method itself uses a terminology close to that of the agile paradigm. It favors iterative development and producing prototypes rapidly. Moreover, it uses concepts favored in the agile paradigm, such as continuous deployment, a feedback loop, and a minimum viable product. IoT-AD claims that it supports iterative development through automation in different phases of IoT application development. The documentation of ELDAMeth covers iterative development with no discussion of a particular paradigm. The documentation on SPLP-IoT and GSEM-IoT does not include any discussion on a process paradigm, not even iterative development.

7.4.11 RIGIDITY OF THE METHOD

Although Ignite's documentation includes many references to other practices and artifacts that can be used with Ignite, it does not include any information on the flexibility of tailoring the process. Ayala et al. (2015) discuss the ability to customize the development process within the scope of SPLP-IoT, but they do not give any information on how to do this. The documentations of IoT-Meth, IoT-AD, ELDAMeth, and GSEM-IoT do not cover process tailoring, which might be needed to meet project-specific needs.

7.4.12 MATURITY OF THE METHOD

Ignite was developed by analyzing best practices and real-world projects. Since Ignite was not used in these projects, a project has been selected to validate it. The project team found Ignite useful in providing a high-level roadmap for implementing an IoT system realizing a business idea. They think that Ignite can be improved further, for instance, to include an approach for keeping track of artifacts during a project. IoT-AD has been evaluated via two case studies in a lab environment. It is reported that IoT-AD, along with IoTSuite, enables us to generate a significant percentage of total application code, resulting in a reduction in development effort, especially for IoT applications involving a large number of devices. It is reported that ELDAMeth has been validated in many case studies

from different application domains, such as distributed information retrieval, mobile e-marketplaces, content delivery infrastructures, and wireless sensor network-based systems (Fortino and Russo, 2012). Some case studies have been conducted to validate SPLP-IoT, and the results have shown that SPLP-IoT leads to autonomous agent systems for IoT systems (Ayala and Amor, 2012; Ayala et al., 2012, 2014). No evaluation or validation cases were reported for IoT-Meth and GSEM-IoT.

7.4.13 DOCUMENTATION OF THE METHOD

Ignite is documented via a book (Slama et al., 2016), which defines the method by providing information from real-world projects. IoT-Meth is only documented through a website (Collins, 2017), which includes a high-level process flow with a presentation giving some information on the method. The main document for IoT-AD is a PhD thesis (Patel, 2014), along with a journal article (Patel and Cassou, 2015). ELDA^{Meth} is a result of a research project, whose outputs are presented on the website* and in some academic articles (Fortino and Russo, 2012; Fortino et al., 2014, 2015). SPLP-IoT is documented by some journal articles (Ayala and Amor, 2012; Ayala et al., 2015) and conference papers (Ayala et al., 2012, 2014). Some conference papers (Zambonelli, 2016, 2017) have been published on GSEM-IoT.

7.4.14 TOOL SUPPORT

Ignite does not propose the use of any specific tool. However, it gives some examples of tools to support development, testing, and so forth, from real-world projects. IoT-AD is supported by an open-source tool called IoTSuite,[†] which aims to make IoT application development easier by supporting the separation of concerns, high-level modeling, and automation. ELDA^{Meth} is supported by a CASE tool called ELDATool, which provides support to developers during the modeling, simulation, and implementation phases for developing ELDA-based distributed agent systems (Fortino and Russo, 2012). The tool was implemented as Eclipse plug-ins. SPLP-IoT discusses some tools that can be used separately to support some steps (configure agent, resolve agent, and model goals illustrated in Figure 7.6) of the process. A software architect can use a tool to check dependencies while configuring an agent to avoid errors that can result from performing this task manually. Moreover, a CVL tool can be used to resolve an agent (combining an agent configuration with an IoT multiagent system architecture to obtain an IoT multiagent system architecture configuration) and produce an IoT multiagent system architecture configuration. Another tool can be used when modeling goals to check the consistency and detect conflicts among agent goals. IoT-Meth and GSEM-IoT do not discuss any specific tool.

7.5 CONCLUSION

As several opportunities are afforded by IoT systems, developing such systems efficiently becomes more important. Generally, developing IoT systems is nontrivial and more complex than traditional software systems, since they encompass many software, hardware, and communication components. Therefore, efficient development of IoT systems requires systematic approaches that come into existence mainly in the form of SDMs. This chapter presents a brief overview of IoT SDMs in the literature and their evaluation based on 14 criteria.

Some IoT SDMs are built based on experience from real-world projects. Ignite and IoT-Meth are two examples of such SDMs emerged from the industry. The SDMs established by the researchers from academia are mainly grounded on previous research in various areas. IoT-AD is built on macroprogramming and inspired by model-driven design. ELDA^{Meth} is based on multiagent system development. SPLP-IoT utilizes the SPLE paradigm, as well as multiagent system development. GSEM-IoT adapts some concepts from traditional software development to the IoT domain.

* ELDA^{Meth} website, <http://eldameth.deis.unical.it/>.

[†] IoTSuite website, <https://github.com/pankeshlinux/IoTSuite/wiki>.

None of the identified IoT SDMs can be considered a complete method that covers all the important phases necessary for developing IoT systems. Ignite presents a more holistic view for developing IoT systems than the rest of the SDMs. It roughly defines a process from an idea to a running IoT system and provides some artifacts. IoT-Meth includes six main steps, which are far from having sufficient detail for guiding the development of an IoT system. IoT-AD, ELDAMeth, SPLP-IoT, and GSEM-IoT focus only the software components of an IoT system.

Validation of these SDMs is an important topic that needs to be addressed. Only Ignite has been validated by one real-world project, but is this enough? Validating these SDMs in real-world projects would allow us to not only assess them, but also to improve them. Moreover, the SDMs should provide adequate documentation covering some basic topics of a method description, such as activities, artifacts, roles, and phases.

REFERENCES

- AIOTI [Alliance for Internet of Things Innovation]. (2016). High Level Architecture (HLA). Release 2.1. AIOTI WG03—IoT Standardisation.
- Ayala, I., and Amor, M. (2012). Self-configuring agents for ambient assisted living applications. *Ubiquitous Computing* 17(6):1159–1169.
- Ayala, I., Amor, M., and Fuentes, L. (2012). An agent platform for self-configuring agents in the Internet of things. In *3rd International Workshop on Infrastructures and Tools for Multi-Agent Systems, ITMAS*, Valencia, Spain, pp. 65–78.
- Ayala, I., Amor, M., and Fuentes, L. (2014). Towards a CVL process to develop agents for the IoT. In *8th International Conference, UCAmI 2014*, Belfast, UK, pp. 304–311.
- Ayala, I., Amor, M., and Fuentes, L. (2015). A software product line process to develop agents for the IoT. *Sensors* 15(7):15640–15660.
- Brinkkemper, S. (1996). Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology* 38(4):275–280.
- Collins, T.A. (2017). Methodology for building the Internet of Things. <http://www.iotmethodology.com/> (accessed January 16, 2017).
- Fortino, G., Guerrieri, A., Russo, W., and Savaglio, C. (2015). Towards a development methodology for smart object-oriented IoT systems: A metamodel approach, pp. 1297–1302. In *IEEE International Conference on Systems, Man, and Cybernetics*, Kowloon, China.
- Fortino, G., Rango, F., and Russo, W. (2014). ELDAMeth design process. In *Handbook on Agent-Oriented Design Processes*, ed. M. Cossentino, V. Hilaire, A. Molesini, and V. Seidita, 115–139. Berlin: Springer.
- Fortino, G., and Russo, W. (2012). ELDAMeth: An agent-oriented methodology for simulation-based prototyping of distributed agent systems. *Information and Software Technology* 54(6):608–624.
- Haugen, Ø., Wąsowski, A., and Czarnecki, K. (2013). CVL: Common Variability Language. In *17th International Software Product Line Conference (SPLC '13)*, New York, pp. 277–277.
- Patel, P. (2014). Enabling high-level application development for the Internet of things. PhD dissertation, Université Pierre et Marie Curie—Paris VI.
- Patel, P., and Cassou, D. (2015). Enabling high-level application development for the Internet of things. *Journal of Systems and Software* 103(C):62–84.
- Pathak, L. M., Bakshi, A., Prasanna, V., and Picco, G. (2007). A compilation framework for macroprogramming networked sensors. *Distributed Computing in Sensor Systems* 4549:189–204.
- Pohl, K., Böckle, G., and van der Linden, F. (2005). *Software Product Line Engineering*. Berlin: Springer-Verlag.
- Slama, D., Puhlmann, F., Morrish, J., and Bhatnagar, R. M. (2016). *Enterprise IoT Strategies and Best Practices for Connected Products and Services*. Sebastopol, CA: O'Reilly Media.
- Zambonelli, F. (2016). Towards a discipline of IoT-oriented software engineering. In *17th Workshop "From Objects to Agents"*, Catania, Italy, July 29–30, pp. 1–7.
- Zambonelli, F. (2017). Key abstractions for IoT-oriented software engineering. *IEEE Software* 34(1):38–45.