# A UML-based Proposal for IoT System Requirements Specification

Gianna Reggio
DIBRIS – Università di Genova, Italy

## ABSTRACT

The paper presents a preliminary version of *IotReq*, a method for the elicitation and specification of the requirements for an IoT-system. The first task suggested by *IotReq* is the modelling of the domain, using the UML and following the service-oriented paradigm, then the goals of the IoT-system to build are elicited and specified, again using the UML and extending the domain model, producing a specification of the functional requirements. *IotReq* also provides preliminary indications for specifying the technological nonfunctional requirements.

A case study, the specification of the requirements for a system to support the Genoa's Science Festival is presented too.

**ACM Reference Format:**
Gianna Reggio. 2018. A UML-based Proposal for IoT System Requirements Specification. In *MiSE'18: MiSE'18:IEEE/ACM 10th International Workshop on Modelling in Software Engineering , May 27, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.1145/3193954.3193956

**CCS Concepts:** Software and its engineering → Requirement analysis

**Keywords:** IoT-system requirement specification, UML, Domain modelling, Service-oriented modelling

## 1 INTRODUCTION

Developing a system based on the Internet of Thing (shortly IoT-system) is a novel task, and unfortunately up to now scarcely supported by software engineering, as stated by the introduction to a 2017 issue of the IEEE Software [5] "*Akin to the mania of 1849 in the hills of California, we're witnessing a software developer's* gold rush *around the Internet of Things (IoT). Neither research nor industry is immune to the fever.*", and later it continues "*Confronted by the wildly diverse and unfamiliar systems of the IoT, many developers are finding themselves unprepared for the challenge. No consolidated set of software engineering best practices for the IoT has emerged. Too often, the landscape resembles the* Wild West*, with unprepared programmers putting together IoT-systems in ad hoc fashion and throwing them out into the market, often poorly tested.*".


*MiSE'18, May 27, 2018, Gothenburg, Sweden*

ACM ISBN 978-1-4503-5735-7/18/05…$15.00
https://doi.org/10.1145/3193954.3193956

An IoT-system presents peculiar features that should be considered to have an effective method for eliciting and specifying its requirements. The domain of an IoT-system, that includes the "things", is composed of interacting dynamic (autonomous) entities of many different types, e.g. cars, car drivers, traffic lights, policemen, parkings, … in the case of a smart system improving the traffic in a city. It will be deployed interspersing its parts (hardware plus software) in the domain, e.g. sensors and actuators on the inanimate things, apps on the mobile phones of the people, servers, …. A very large numbers of hardware and software technologies having quite different characteristics may be used to build an IoT-system (e.g. RFID, beacons, mobile phone sensors, …).

In this paper we consider the elicitation and specification of the requirements for an ioT-system, a task for which an extremely small number of proposal can be found in the literature (see Sect. 5), whereas our experience interacting with local companies witness unstructured textual documents without a clear distinction between functional requirements and directives about the hardware/software IoT-related technologies to use. Sometimes, the technologies to use are considered before to have a clear view of the problem to solve, and in some extreme cases they may obfuscate alternative convenient solutions based on less innovative technologies.

Following the suggestion of [5] "*In short, past software engineering techniques can be harnessed and adapted to the challenges of today's IoT.*", we have worked out a preliminary proposal of a method, *IotReq*, for eliciting and specifying the requirements of an IoT-system combining in an ad hoc fashion classical software engineering techniques, such as the preliminary modelling of the system domain and the goal-oriented requirements [10], and the service-oriented UML modelling method introduced in [7].

The first task suggested by *IotReq* is modelling the domain, using the UML and following the service-oriented method introduced in [7], then the goals of the IoT-system to build are elicited and specified, again using the UML and extending the domain model, producing a specification of the functional requirements. *IotReq* provides also preliminary indications for collecting and specify the nonfunctional requirements concerning the technologies to use.

In some sense, *IotReq* offers to a developer in the IoT context something of equivalent to what it is nowadays a standard approach: the combination of "UML conceptual modeling" plus "requirement specification expressed using use cases, specified again with UML".

We have applied *IotReq* to a realistic case study: a system supporting the Genoa's Science festival, used as a running example in the paper, and as a result we have been able to capture and specify its requirement without any overspecification.

Sect. 2 presents an overview of *IotReq*; Sect. 3 and 4 detail the domain modelling, and the requirement specification; then the related works are reported in Sect. 5, and Sect. 6 concludes the paper and hints to the future work.

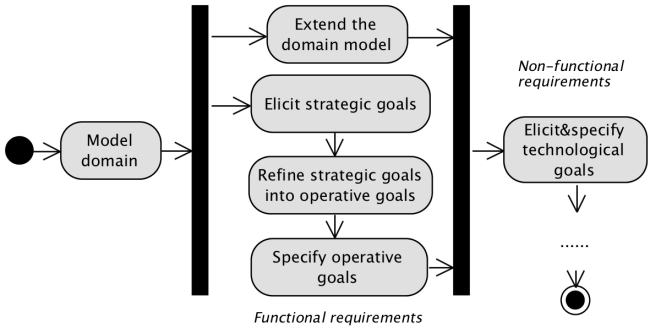


**Figure 1: *IotReq* Overview**

## 2 OVERVIEW OF *IOTREQ*

*IotReq* is a method to support the elicitation and the specification of the requirements for an IoT-system, denoted in what follows by IoT-Sys, and intended as in [4, page 74].

Fig. 1 summarizes the *IotReq* tasks.

IoT-Sys will be built to act over a fragment of the real world to achieve some desired effects (e.g. to improve the traffic in the city downtown), and at the end the various (hardware and software) components of IoT-Sys will be deployed on that fragment (e.g. sensors detecting the passing of the cars, videocameras, and apps on the car driver mobiles in the case of the traffic system; RFID devices attached to the tickets or to visitor's lapel pins, beacons, and again apps running on mobile phones in the case of the Genoa's Science Festival). In this paper we name *domain (of the IoT-Sys)* that fragment of the real world.

The first task of *IotReq* is then modelling the domain using the UML, and following the service oriented method introduced in [7]. The domain of an IoT-system is in general quite complex, made by dynamic entities of different types interacting among them, and thus it is very important for the analyst to understand it before to look for the requirements.

Later, the analyst must look for the ultimate goals of IoT-Sys, that we name strategic goals; then they will be decomposed into operative goals, that will be specified using the UML by extending the domain model. That amounts to specifying the functional requirements for IoT-Sys. At the end, also the nonfunctional requirements should be investigated, those concerning the choice of which technological means to use to realize the IoT-Sys will be expressed by further decomposing the operative goals into technological goals.

To present *IotReq* we use as running example a realistic case study: specifying the requirements for a system to support the Genoa's science festival (Festival della Scienza, shortly FdS).

FdS consists of a large number of events of different types that take place in different locations in the Genoa's downtown. Some of the events are free, other needs to be booked in advance, furthermore events may be replicated several times at different dates and time (e.g. guided visits and labs). It is possible to buy a ticket for the whole festival (season ticket) or for a single day.

IoT seems a natural choice for a system supporting the working of FdS; indeed, for example, it may make possible: – to provide the visitors with information depending on their position inside the festival area, – to collect the rating of the events only by who truly attended them, – to discover which paths the visitors followed moving between the various festival locations.

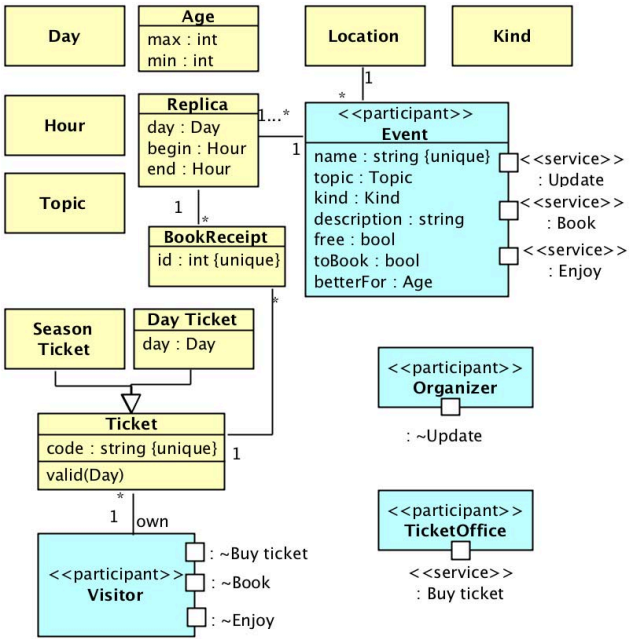


**Figure 2: FdS Domain Model: Static View**

## 3 *IOTREQ*: DOMAIN MODELLING

*IotReq* requires to model the domain of IoT-Sys (the system to be built) following the service-oriented paradigm, and using UML [9] as proposed by [7]. That it is achieved using a UML profile providing concepts as *service*, *participant* that interact with other participants by providing and using services, *service architecture*. i.e. the structure of a system in terms of participants and of the relationships between who provides and who uses the various services.

Thus, the domain of an IoT-Sys will be modelled in terms of participants providing and using services.

Technically, a *model of a domain* consists of: – a *static view*, a UML class diagram introducing the classes modelling the participants and the *objects* (passive entities manipulated by the participants); – the *models of the services* they provide and use; – and a *service architecture*.

The complete model of the FdS domain is in [2].

*Static View.* The classes in the static view typing the participants are stereotyped by ≪participant≫. A participant class may have ports stereotyped by ≪service≫ and typed with a service interface (see next Paragraph "Service model") to represent the provided services, and ports typed by the conjugate of a service interface (whose name is prefixed by ∼, again see Paragraph "Service model") to represent the used services.

Fig. 2 presents a slightly simplified version of the static view of the FdS Domain Model. The main entities of FdS are the events of the festival, the visitors, the festival organizer, and the ticket office (classes stereotyped by ≪participant≫). Fig. 2 shows that an event

is characterized by a unique name, a kind, a topic and a description, and it takes place in a specific location; moreover it may require buying a ticket or be free, it may require a booking, and it is suitable for a specific age. Each event will be replicated at least once, and each replica will happen on a given day and time. There are two kinds of tickets, one valid for the whole festival (season ticket), and another one allowing access to all events in a specific day.

The Event class provides the services for being booked, updated and enjoyed, whereas Visitor uses the services for booking and enjoying an event, and for buying a ticket (provided by the Ticket Office class).

Detailing the static view (e.g. adding invariants, attributes, operations, associations, and definitions of the operations) it is possible to model all the relevant aspects of the domain; for example, the fact that only non-free events may be booked could be formalized by an invariant constraint, whereas the multiplicity constraints on the associations allow, e.g. to state that a booking is relative to a specific replica and a specific ticket. The fully detailed static view of the FdS Domain Model is reported in [2].

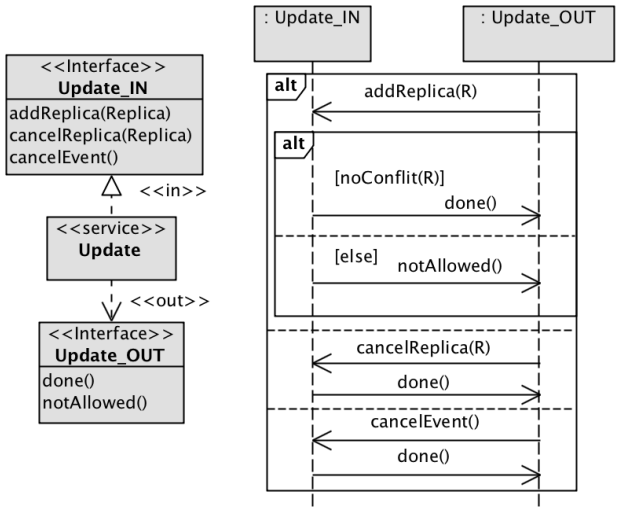


**Figure 3: Update service: model**

*Service model.* A model of a service consists of the *interface*, and of the *contract+semantics*.

A *service interface* is a UML class stereotyped ≪service≫ and named as the service itself. It should realize and use two UML interfaces, defining the in/out-messages of the service (in messages are those sent by the service user and received by the service provider, whereas, vice versa, the out messages are those sent by the service provider and received by the service user), named respectively S_IN and S_OUT, if the service is named S. The operations of the interfaces define the various messages. The *realization* relationship is represented by a dashed arrow with closed head stereotyped by ≪in≫, whereas the *usage* relationship is represented by a dashed arrow with open head stereotyped by ≪out≫.

The service contract+semantics is a set of sequence diagrams having exactly two lifelines typed respectively by the interfaces S_IN and S_OUT, and thus the sequence diagram messages are labelled by operations of these interfaces, that, as said before, represent the service messages. The sequences of messages expressed by the sequence diagrams define the service contract (i.e. the protocol to follow to provide and use it), whereas the guards and the execution specifications [9, 17.2.4.4] define the semantics of the service, that is the provided value.

A *conjugate* service interface is suggested as a mechanism to connect the using participant and the providing participant. Each service interface has one conjugate service interface that is named by the name of the corresponding service interface prefixed with "∼", and it is defined transforming the in-messages into out-messages, and similarly the out-messages into in-messages, i.e. the realized interface becomes the used one and vice versa.

In the FdS Domain Model the Update service describes at conceptual level how the events may be modified, and its model is shown in Fig. 3. Its interface consists of three in-messages modelling the fact that replicas may be added and cancelled and that events may be cancelled (Update_IN), whereas the out messages correspond to the fact that an update may be done or denied (Update_OUT). The contract+semantics states that cancellations are done in any case also if there were booked, whereas new replicas are added only if they do not conflict with existing ones.

The models of the other services, reported in [2], describe the other behavioural features of the domain of FdS in a modular way: each service cover a specific feature (e.g. booking an event).

*Service architecture.* The service architecture is a UML composite structure diagram including a structured class named as the domain to model itself. The roles for the entities composing the domain will be represented as parts[1] of the domain class typed by participant classes, they will have a multiplicity, and possible a name. A port of a part typed by a service interface must be connected to another port typed by the conjugate interface, and a port typed by the conjugate of a service interface must be connected to a port typed by that service interface; summarizing each provided service must be used by at least a participant, and each used service must be provided by another participant. The service architecture will represent the structure (architecture) of the domain.

Fig. 4 shows the service architecture of the FdS Domain Model: there will be exactly one organizer and one ticket office, and any number of events and visitors. The visitors use the services Enjoy and Book provided by the events, and the service Buy ticket provided by the ticket office; the organizer uses the upadate service provided by the events.

## 4 *IOTREQ*: REQUIREMENT SPECIFICATION

Remind that in what follows IoT-Sys denotes the IoT system to be built over a domain denoted by Domain.

To elicit and specify the requirements for IoT-Sys we follow the goal paradigm: "*Goals are desired system properties that have been expressed by some stakeholder(s).*" from [10]. Since an IoT-system is embedded or better strictly interweaved with the domain (e.g. think of an IoT-system made of hardware parts such as sensors, videocameras, but also of apps running on the car driver mobiles, and software making predictive analysis on the collected data sparse

[1]Part is the UML constructs to represent the subcomponents of a structured class.

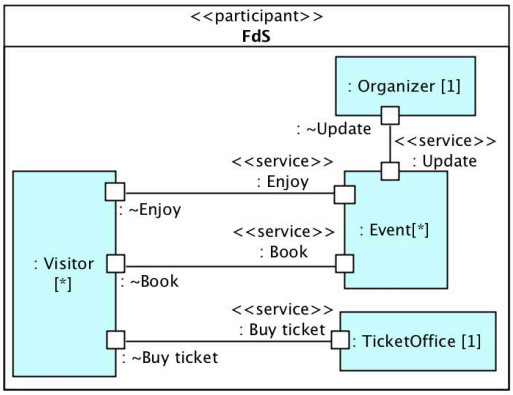


**Figure 4: FdS Domain Model: service architecture**

in a city downtown), it is convenient to consider the "desired system properties" as properties of the domain that the introduction of the system should get (e.g. "no truck should be moving or parked at less than half km from any crowded place"). Thus a goal should correspond to a property on the domain, that will be then specified using the UML on the model of the domain (see Sect. 3).

A goal may be, for example, – an invariant to hold (e.g. "no more than 5 persons may be inside a room"), – a condition or some activities in the domain must trigger some other activities (e.g. "if a gasoline pump becomes empty, then the car owners must be informed by a message containing the list of the nearest pumps"), – additional features that must enrich the domain (e.g. car drivers should be able to see if a parking is full, and to book a place in a parking in advance), – prohibited behaviour (e.g. visitors with luggage cannot enter in the event locations), – and even questions to be answered on the domain and its behaviour (e.g. what is the average of the number of events enjoyed by visitors with day/season ticket?).

In the case of FdS sensible goals may: "Send a remind to the visitors one hour before a booked event", "When a visitor leaves an event, s(he) must be asked to evaluate it" (of type triggered behaviour), and "A visitor may get information on the buses to take to reach an event location" (of type additional behaviour).

If the requirements elicitation starts asking the stakeholders to list the main goals of the IoT-Sys, that are the basic reasons to build it, you will get goals so high-level or vague that it is impossible to express them precisely as properties of the domain, for example "traffic jam should be reduced" (no way to precisely define what means to reduce), "break-ins in the house should be prevented", but they are in general what motivate the development of IoT-Sys. We name these goals *strategic* as in [10]. *IotReq* proposes to start the requirement elicitation procedure looking for the ultimate reasons that motivate the IoT-Sys's development, also if they may be vague or too abstract and thus not verifiable, or also not satisfiable as a whole[2]; then such goals should be refined in terms of subgoals that could be expressed as properties or features of the domain (called *operative goals*): they represent the functional requirements for IoT-Sys (the IoT-system to build).

[2]Clearly, they cannot be considered true requirements, since a requirement should be satisfiable and verifiable.

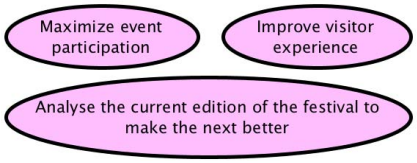


**Figure 5: FdS Goal View: Strategic goals**

On the other hand, functional requirements should not encompass implementative choices and details, for example the operative goals should not state that some messages are sent using SMS/email/visualized by an app on the mobile phones, or that the number of cars in a street section is determined by analysing of the imagines of a camera/by sensors on the streets. The non-functional requirements concerning the choice of technologies to use for realizing IoT-Sys may be expressed by further refining the operative goals by means of goals stating which technologies to use to reach them (*technological goals*). Thus, *IotReq* asks first to look for the strategic goals, then to refine them into operative goals, and finally to consider the technological aspects, by further refining the operative goals into technological ones (see Fig. 1).

Summarizing, a *requirement specification for an IoT-system* will consist of: – a domain model as presented in Sect. 3, – a *goal view* summarizing the goals and their mutual relationships, – and the goals' specifications expressed using UML diagrams and constructs.

## 4.1 The Goal View

The goals are represented by UML use cases without actors stereotyped by ≪goal≫, and are collected in a use case diagram named *goal view*. Goals may be related by:
– UML dependency [9] (represented by a dashed arrow with open head): $G_1$ depends on $G_2$ iff the realization of $G_2$ is necessary to the realization of $G_1$;
– dependency stereotyped by ≪or≫: G or-depends on either $G_1$, …, $G_n$ iff the realization of either $G_1$, …, or $G_n$ is necessary to the realization of G.

The goal name should summarize the required property, whereas an attached optional note may add the needed details using the natural language. Strategic goal are represented by ellipses drawn with a thick line and no UML specification will be attached to them; whereas operative goal or just goal are represented by ellipses drawn with a thin line, and must be specified using the UML in terms of the domain.

Fig. 5 shows the portion of the goal view of the FdS system containing the strategic goals (the complete goal view is reported in [3]), that are quite generic and not verifiable, but they are the starting point to elicit and specify the true functional requirements. Since the FdS events are organized by volunteers, it is important that each event get a high attendance, because this is the only kind of reward for them; and this is the motivation for the goal "Maximize event participation ".

Strategic goals must be refined by decomposing them in subgoals related by dependency/or-dependency. At the end of the refinement task all non-strategic goals should be verifiable and realizable, and expressible as properties of the domain: they represent the functional requirement for IoT-Sys, and are called *operative* goals. At


**Figure 6: Maximize event participation goal decomposition**

this point each operative goal must be specified by using the UML diagrams and constructs possibly extending the domain model.

Fig. 6 shows the refinement of the strategic goal Maximize event participation of FdS. First, it has been decomposed in an operative subgoal and in a strategic subgoal (noted by the thick line), that was later decomposed in two operative subgoals. The three operative subgoals must be specified using the UML, and such specifications will allow to fix all the detail and to avoid any ambiguities and undefined aspects.

## 4.2 Goal specification

Before to suggest which UML constructs/diagrams to use and how to specify the meaning of a goal, we recall that the atomic activities of the entities composing the domain (participants) are modelled by the sending/receiving of the messages composing the interfaces of the services that they provide and use (atomic interactions with other participants), and by self calls of operations of their classes without return type (atomic private activities); for example the visitors (participants of the FdS Domain Model typed by the Visitor class) may book an event and buy a ticket.

The first step to specify a goal is to determine which participants and objects of the the domain are involved, or better which roles they will play (e.g. to specify the goal of the FdS system Remind visitors their bookings we need to refer to a generic visitor, that is a role typed by the Visitor class). The roles of a goal will be represented by a list of variables typed by classes appearing in the static view of the domain model, and we will write G($x_1$: $C_1$, …, $x_n$: $C_n$) to express that the G goal involves the roles $x_1$, …, $x_n$.

The method proposes a list of patterns, reported in Table 1, for the specification using the UML of a goal G($x_1$: $C_1$, …, $x_n$: $C_n$), that obviously cover only the most frequent cases, where
– cond($x_1$,…,$x_n$) denotes a boolean OCL expression [6], where $x_1$, …, $x_n$ may appear as free variables;
– sd1($x_1$,…,$x_n$) and sd2($x_1$,…,$x_n$) are sequence diagrams whose lifelines are typed by $C_i$ and named by $x_i$ (with $1 \leq i \leq n$ and $C_i$ stereotyped by ≪participant≫), whose messages are labelled by calls of operations belonging to the interfaces of the services part of the domain model, and whose execution specifications (see [9, 17.2.4.4]) are labelled by calls of operations without return type of the participant classes;
– ad1($x_1$,…,$x_n$) and ad2($x_1$,…,$x_n$) are activity diagrams whose actions are calls of either operations belonging to the interfaces of the services part of the domain model or of operations without return type of the participant classes, and whose swimnlanes (if any) are labelled by $x_i$: $c_i$ (with $1 \leq i \leq n$ and $C_i$ stereotyped by ≪participant≫).

- **Invariant**
  cond($x_1$,…,$x_n$)
- **Triggered behaviour**

| Trigger | Effect |
|---|---|
| cond($x_1$,…,$x_n$) | sd2($x_1$,…,$x_n$) |
| sd1($x_1$,…,$x_n$) | sd2($x_1$,…,$x_n$) |
| cond($x_1$,…,$x_n$) | ad2($x_1$,…,$x_n$) |
| ad1($x_1$,…,$x_n$) | ad2($x_1$,…,$x_n$) |

- **Unwanted behaviour**
  sd1($x_1$,…,$x_n$) or ad1($x_1$,…,$x_n$)
- **Restricted behaviour**

| Activity | only if |
|---|---|
| sd1($x_1$,…,$x_n$) | cond($x_1$,…,$x_n$) |
| ad1($x_1$,…,$x_n$) | cond($x_1$,…,$x_n$) |

- **Additional behaviour**
  sd1($x_1$,…,$x_n$) or ad1($x_1$,…,$x_n$)

**Table 1: Patterns for UML based goal specification**

• *Invariant* A goal requiring that some static property on the domain (i.e. not concerning the dynamic behaviour of the participants) should be always true.
• *Triggered behaviour* A goal requiring that when some static condition become true/something happen in the domain (trigger), then some specific activities must be done (effect). All operative goals in Fig. 6 conform to this case, see for example Fig. 8.
• *Unwanted behaviour* A goal requiring that some specified behaviour should be made impossible by the IoT-system, may be expressed by either a sequence diagram or an activity diagram.
• *Restricted behaviour* A goal requiring that some specific activities may be performed only in a restricted number of cases, defined by a condition on the goal roles.
• *Additional behaviour* A goal requiring that some specific new activities may be performed, for example in the FdS case the goal Evaluate festival is of this type.

It may happen that to specify an operative goal it is necessary to interact with entities not included in the domain. e.g. "if an intruder is detected in a room, then the police should be alerted", but the police was not part of the home domain. In these cases, the domain should be extended, by adding further participants with their services and objects, so that any entity needed to phrase the operative goals with the UML is available in the domain. For example in the FdS case we added the participant Bus company to allow to specify that the IoT-system to be built should interact with it to support the buying of the bus tickets.

Similarly, the entities already present in the domain may need to be extended by adding new attributes, operations and provided or used services, as well as new classes needed to type them. Again in the FdS case we added new services, such as Notify (to abstractly model the fact that the visitors should be notified various messages) and Give position (to abstractly model the fact that the position of the visitors should be detected and made available). In Fig. 7 we present a fragment (the complete one is shown in [3]) of the extended static view with all the additions needed to specify the subgoals of Maximize event participation, see Fig. 6.

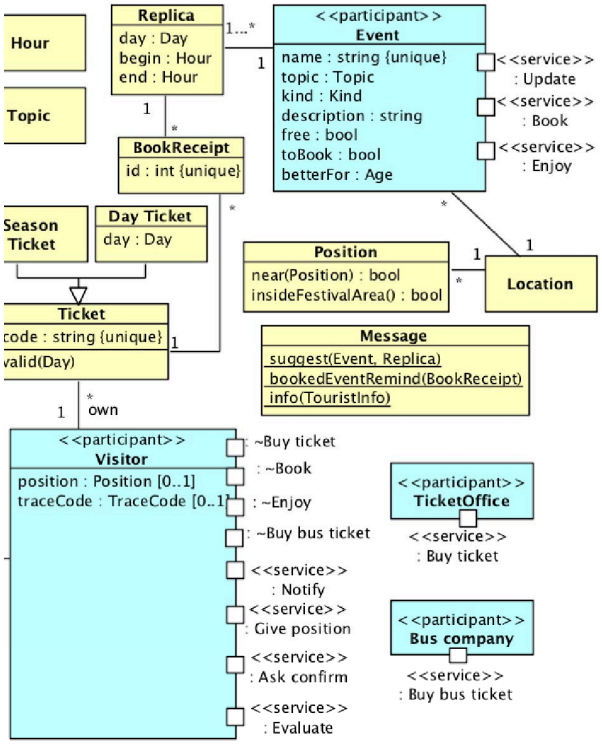


**Figure 7: FdS requirement specification: fragment of the Static view of the extended domain model**

The specification of the Suggest next events goal is reported in Fig. 8 (it is of kind triggered behaviour): any time a visitor is near (what does it means "to be near" is left to the definition of the operation near of class Position) one hour before the start of an event that it is not fully booked s(he) will receive a notification. An overkill variant of this requirement may be proposing to a visitor only events that s(he) may like based on the-fly analysis of which s(he) has enjoyed previously or booked and on preferences declared when buying a ticket; but this is too much for the specific case, where the number of the events is less than 100.

Lost messages (those with the arrow reaching a black circle instead of a lifeline) are provided by the UML for the cases where the target of a message in a sequence diagram is not fixed, similarly for the found messages (those starting from a black circle). Lost and found messages are used in the sequence diagrams representing the trigger and the effect of a goal specification to avoid that the specification of a goal results in determining the architecture of the system to be built (i.e. encompassing implementative details in the requirements). In the case in Fig. 8 using lost/found messages we were not forced to say which hardware/software components of the system will receive the position and which will send the notification, but the intended meaning of the goal is clear: IoT-Sys should make such sequence of message happen in such cases.

Fig. 9 shows instead the specification of the Ask for booking confirmation goal, again it is of kind triggered behaviour, but now the trigger is a condition. To specify this goal the Visitor class provides now a new service Ask confirm (see [3]).

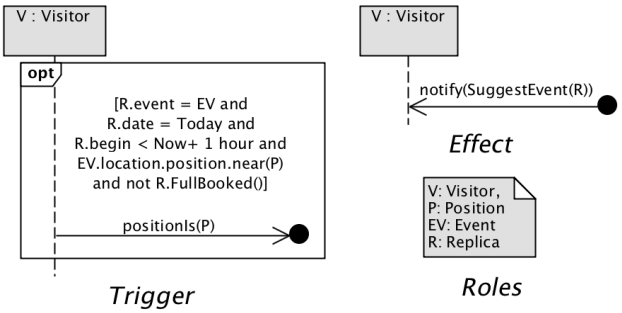


**Figure 8: Goal Suggest next events specification**

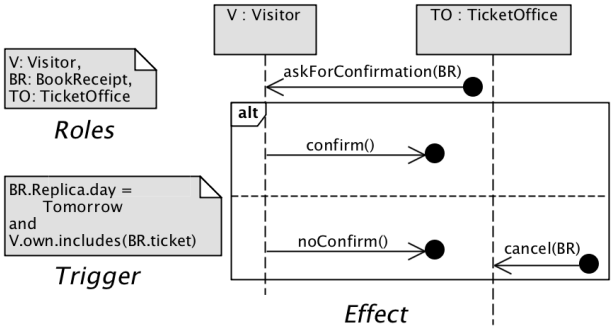


**Figure 9: Goal Ask for booking confirmation specification**

The specifications of the other goals of the IoT-system for FdS can be found in [3]. Notice that also which data produced by the FdS current edition must be collected is specified, and also which are the required analysis (see the goal "Analyse the current edition of the festival to make the next better", and the specifications of its operative subgoals).

Fig. 10 presents the updated serviced architecture.

Since none of the existing participants seems the right choice as provider/user of some of the new services (e.g. Notify and Give position), we should introduce new participants, but there is not an obvious way to determine them. Leaving such providers/users undetermined the task of specifying the goals is not hindered, and no information relevant for the next development phases is lost. Similarly to the lost and found mechanism for messages, *IotReq* introduces the lost and found services. A lost service is a provided service for which the users are not defined, similarly a found service is a used service whose provider is not determined. Technically, they are represented by ports typed by a service interface or a conjugate of a service interface stereotyped by ≪undetermined≫ and represented by the black box icon.

In Fig. 10 the participant Bus company with the service Buy bus ticket has been added to specify the goal *Improve visitor experience*. The four new services provided by the Visitor class are lost services represented by the black box, as well as Record, whereas Analyse is a found service.

An operative goal may be complemented with the usual information associated with use case, such as stakeholders, priority and frequency, as for the use cases, that will help in deciding in which order the goals should be considered in the development activities.

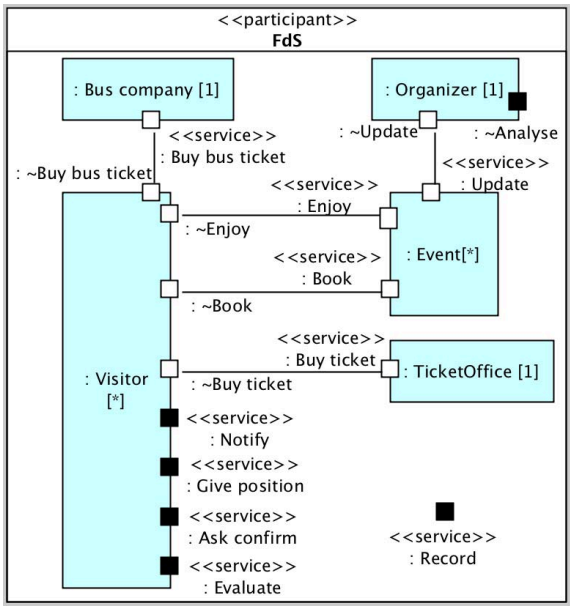


**Figure 10: FdS requirement specification: Service architecture of the extended domain model**

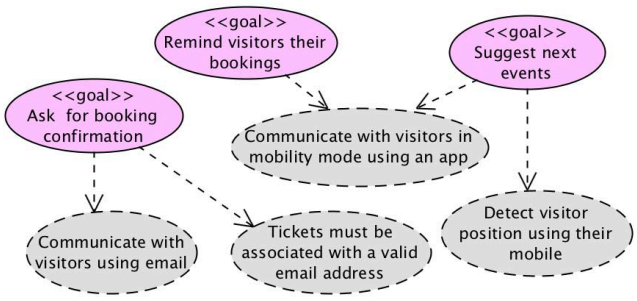


**Figure 11: Goal Maximize event participation: Technological goals decomposition**

## 4.3 Nonfunctional requirements

The preliminary version of *IotReq* presented in this paper provides also some hints to tackle just one kind of nonfunctional requirements for an IoT-system, being aware that defining all the types of the nonfunctional requirements for IoT-systems needs a thorough investigation, that will the subject of future work.

In the case of IoT-systems there are many different possibilities for the choices of the technologies to use to build them, both at the hardware level (e.g. sensors, beacons, RFID, mobile devices, videocameras, appliances) and software (e.g. rest services, various protocols) with very different characteristics (e.g. concerning cost, installation, maintenance), so the "implementation requirements" are more complex to elicit and specify than for classical software. Furthermore, the use of the above mentioned technologies is what characterises an IoT-system (see [4]), thus they should be explicitly taken into account in the requirement elicitation phase. Our method proposes to present the implementation requirements for an IoT-system by using the *technological goals*, that are goals refining the operative goals stating which technological means must be used to realize the refined goals; obviously, the considered technologies should be those related to the IoT.

The technological goals will be represented by dashed ellipses, and should be summarized in a goal view containing also the operative goals. We suggest to consider different options when looking for the supporting technologies, and to represent them by using the or-dependency among goals, then it will be possible to perform an analysis of the various possibilities helping the analyst to select a coherent set of technologies.

Fig. 11 presents the technological goals corresponding to the strategic goal Maximize event participation, whereas all the others are listed in [3]. Notice that also a non-IoT technology is considered (i.e. email), allowing to reach also visitors that decide to not use the Festival's app. A different choice, based on lapel pins equipped with RFID to be obtained after buying a ticket and on beacons distributed in all the festival locations was ruled out for the costs, too high for the FdS budget.

## 4.4 The Genoa's Science Festival Case Study

The Genoa's Science Festival case study was prompted by a student project requiring to build "a Mobile Cloud Computing system for monitoring, analysing and visualizing the data about the FdS visitors" proposed in a course on IoT-technologies and development tools. We tried to specify the requirements for such system using *IotReq*, and the result was satisfactory, indeed all relevant aspect of the domain were modelled, and any relevant requirement was specified. The resulting model and specification are quite easy to produce for a typical UML user, since *IotReq* provide strict guidelines on which UML constructs to use and how. Moreover, they are also quite easy to understand again for a typical UML user, for example the goal specifications produced following the suggested pattern are easy to grasp.

Furthermore, the service-orientation provides a nice modularization means for the interactions among the entities composing the domain, helping to build and maintain the various models, and again understand them. Consider Fig. 4, it summarizes the behavioural aspects of the domain, then the various service models add the details. Assume that the event updating is no more relevant, it is very simple to modify the model: just drop a participant and a service. Using other UML constructs to represent the behavioural aspects of a domain, for example associating a state machine with each participant class, will result in much more complex models and surely less modular (to drop updating in this case means to examine a state machine looking for the part related with updating and then eliminate it).

Using *IotReq* the focus of the system has been better understood, making it more suitable to provide value to the festival organizer, since one was lead to think to the functions of the system abstracting from the available technological solutions. For example, monitoring was found irrelevant, and the data analysis were required to be performed off line at the end of the festival (as a consequence, the system should be simpler to implement). In the student project one of the main features was to track the movement of the visitors inside the festival area, but analysing such paths cannot provide a big help to the organizers, e.g. to know that people once reach

a location then attend all event taking place there cannot help to better organize the next edition of the festival (thus the priority of the corresponding goal is very low). Instead, some of the most useful analysis can be done collecting data on the ticket owners and recording the events attended using the tickets, allowing to understand the typology of the visitors, the most/least attended events, and the relationships between the visitor and the events characteristics. Such results may be useful to select the events for the next edition, and to better advertise the festival.

## 5 RELATED WORK

Requirement engineering for IoT-systems, and in general software engineering for such systems, is a very recent field, indeed one of the most valuable reference is the first issue of the 2017 of the IEEE Software journal [11]. One of the included papers [12] is the most interesting, since it tackles the topic of requirements for IoT-systems. This paper proposes a conceptual framework for capturing and presenting the requirements. First it is required to find the actors in the system (global and local managers, and users), and then to look for the associated *policies* (corresponding to our goal of kind invariants), *goals* (corresponding to our goals of kind triggered behaviour) and *functions* defined by "*Functions define the sensing, computing, or actuating capabilities of individual things or a group of things, or specific resources that are to be made available. Functions are typically accessible as services*" [12]. In our proposal their role is played by the services provided by the domain participants. At conceptual level our current proposal is simpler, considering only non-further classified participants providing and using services, and non-further classified goals. As a future work we plan to investigate whether a similar or a different classification may better support the requirement elicitation and specification. However, [12] does not tackle the requirement specification issue.

In the literature there are various proposal for using the UML, or better UML profiles, for supporting the development of IoT-systems, e.g. [8], but only the design and implementation phases are considered not the requirement one. Instead, [1] suggests to use a profile of SysML (that in turn is a UML profile) for modelling IoT-systems, and proposes a development methodology, covering not only the design phase, indeed the authors suggest to specify both the requirements for the IoT-app (not clear if it is the whole IoT-system or a part of it) as a black box, and for the used devices. [1] differs from our method since does not require an explicit domain model, consider the IoT-system as a black-box and not interspersed on the domain, and the requirements are textually specified.

## 6 CONCLUSIONS AND FUTURE WORK

We have presented a preliminary version of a method, *IotReq*, that combines "service-oriented" UML modeling, and simple well-known software engineering practices, to support the elicitation and the specification of the requirements for an IoT-system. The analyst first must understand and model using the UML the domain of the IoT-system to build, then s(he) is driven to look for the ultimate (strategic) goals of the system, later to decompose them into operative goals, that can be quite precisely specified to avoid ambiguities and incompleteness, again using the UML. Finally, s(he) should look for the nonfunctional technological goals.

The method has been applied to a realistic case study: FdS (the Genoa's Science Festival), showing that it allows to precisely specify its requirements, separating the functional ones from those concerning the technologies to use. And the resulting specification is quite readable. However, we are currently working on two other case studies, with quite different characteristics, provided by local companies, to further validate *IotReq*.

The current version of *IotReq* is quite simple, and thus can for example taught in few hours to someone familiar with the UML and the service paradigm, and it just allows to precisely specify the requirements for an IoT-system. It may, and will, extended:
– as it has been done in [7], we plan to precisely define the form of the UML models used for the domain and the requirement specification by means of constraints to help avoid common mistakes, and guarantee by construction a certain level of quality of the produced models, and a tool based on model transformations may check that are satisfied;
– using again model transformations the requirement specification may be transformed in inputs for goals' analysis tool, e.g. to automatically guarantee the absence of conflicts among the goals;
– other "classical" software engineering practices may be incorporated, e.g. to derive tests from the requirements.

Providing classification schemas for participants, services and goals (e.g. internal/external, i.e. out of the control of the developer as the Bus company of FdS, list of all the possible technological goals) are obviously useful, we plan to add them after some empirical investigations on the current practice in IoT-development.

## ACKNOWLEDGMENTS

I would like to thank the reviewers for their carefully reading and useful suggestions.

## REFERENCES

[1] B. Costa, P. F. Pires, and F. C. Delicato. 2016. Modeling IoT Applications with SysML4IoT. In *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 157–164. https://doi.org/10.1109/SEAA.2016.19
[2] G. Reggio. 2018. Genoa's Science Festival Domain Model. (2018). Available at `sepl.dibris.unige.it/2017-GenovaScienceFestivalCaseStudy.php`.
[3] G. Reggio. 2018. Genoa's Science Festival Requirement Specification. (2018). Available at `sepl.dibris.unige.it/2017-GenovaScienceFestivalCaseStudy.php`.
[4] IEEE Internet Initiative. 2015. Towards a definition of the Internet of Things (IoT). (2015). Available at `iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf`.
[5] X. Larrucea, A. Combelles, J. Favaro, and K. Taneja. 2017. Software Engineering for the Internet of Things. *IEEE Software* 34, 1 (2017), 24–28. https://doi.org/doi.ieeecomputersociety.org/10.1109/MS.2017.28
[6] OMG. 2003. *UML 2.0 OCL Specification.*
[7] G. Reggio, M. Leotta, D. Clerissi, and F. Ricca. 2017. Service-oriented domain and business process modelling. In *Proceedings of the Symposium on Applied Computing, SAC 2017.*, Ahmed Seffah, Birgit Penzenstadler, Carina Alves, and Xin Peng (Eds.). ACM, 751–758. https://doi.org/10.1145/3019612.3019621
[8] Kleanthis Thramboulidis and Foivos Christoulakis. 2016. UML4IoT – A UML-based Approach to Exploit IoT in Cyber-physical Manufacturing Systems. *Comput. Ind.* 82, C (2016), 259–272.
[9] UML Revision Task Force. 2015. *OMG UML, V2.5.*
[10] Various. 2007. *A KAOS Tutorial.* Available at `www.objectiver.com/fileadmin/download/documents/KaosTutorial.pdf`.
[11] Various. 2017. Software Engineering for the Internet of Things. *IEEE Software* 34, 1 (2017).
[12] F. Zambonelli. 2017. Key Abstractions for IoT-Oriented Software Engineering. *IEEE Software* 34, 1 (2017), 38–45. https://doi.org/10.1109/MS.2017.3