

# Multi-Level SLA Specification Language for IoT Applications

Nouredine Staifi<sup>1</sup>, Meriem Belguidoum<sup>1</sup>

<sup>1</sup>LIRE Laboratory, University of Constantine 2, Algeria

## Abstract

Service level agreement (SLA) is a formal contract between a service provider and a service consumer, used to guarantee the achievement of Quality of Service (QoS) expectations. An IoT application is made up of several layers in which several devices are used, the main challenges can be summarized into two points: (1) how to describe the SLA terms, such as QoS properties, service levels, penalties in SLA violation, and (2) how to integrate an SLA into all the application layers. For that, SLAs must be specified more expressively and encompass concepts and constraints related to the multi-layered nature of IoT applications and user preference-aware. However, almost all existing SLA specifications are unable to cover all these requirements. We believe that an SLA can be represented in a domain-specific language (DSL) to facilitate the expression of SLA terms, automate the negotiation, and adapt the provided services according to these terms. Therefore, in this paper, we propose a DSL called ML-SLA-IoT, which is a multi-level SLA specification language with customisable, parametrised and dynamic management, intended for IoT applications. ML-SLA-IoT is more advantageous than others languages in terms of specification of user preferences, the use of microservices and the ML-SLA concepts.

## Keywords

Service Level Agreement, Quality of Service, SLA specification, Domain-specific language, Internet of Things, Multi-level SLA, User preference-aware, Microservices.

## 1. Introduction

Services are marketed under formal contracts known as Service Level Agreements (SLA) which is a formal contract between a service provider and a service consumer. It formally specifies the provided services, the obligations of each party and the applicable penalties in the case of non-compliance with the contract [1]. The SLA aims to meet customer's Quality of Service (QoS) expectations and allows each party to respect its commitment, and in the case of conflict, it improves the understanding aspect between the involved parties. The consumer uses SLA as a legal description of what a provider has promised to provide. In turn, the provider uses it as a record of what to be delivered [2].

User expectations must be provided with guaranteed QoS levels, in which requirements are formally specified in SLA. Guaranteeing QoS is a difficult task to achieve [3], e.g. IoT application systems are generally made up of several layers involving multiple heterogeneous hardware and software resources, multiple data types from digital and wearable sensors and multi QoS levels

---


*Tunisian Algerian Conference on Applied Computing (TACC 2021), December 18–20, 2021, Tabarka, Tunisia*

✉ noureddine.staifi@univ-constantine2.dz (N. Staifi); meriem.belguidoum@univ-constantine2.dz (M. Belguidoum)

🆔 0000-0001-9965-9785 (N. Staifi); 0000-0002-2936-6810 (M. Belguidoum)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

from the lowest to the highest one. Providing QoS guarantees requires the technical ability to ensure that their QoS requirements are met on each IoT application system layer.

There are three types of SLAs [4], firstly, Customer-based SLA is an agreement that puts all the customer's requirements and expectations into a single document, this means that all the provided services will be described and contained in one SLA. Secondly, Service-based SLA is an agreement in which the provider lists services in a service catalogue, each service has its own fixed SLA parameters, which do not take into account user preferences. Finally, multi-level SLA (ML-SLA) is an agreement that is classified into different stages, it is customized according to the end-user requirements, it allows the user to integrate several conditions into the same system to create a more suitable service. ML-SLA is a solution to solve the problems related to: (1) the integration of contracts into the entire multilayer architecture, and (2) the flexibility limitation resulting from the static nature of traditional SLA approaches in terms of QoS and pricing.

Mostly, SLAs were written using natural language and the agreement compliance verification used to be done manually. Some solutions have been proposed to facilitate this process using SLA models, nevertheless, those solutions lead to new problems in specifying different service levels for different customers [5]. To resolve these problems, it is necessary to automate the process in which different SLAs are described, provided and observed flexibly. To simplify the contractual process for the involved parties and optimise time and costs, researchers from the service provider community have developed several SLA specification languages [5]. These languages are important for clarifying expectations and quantifying them in measurable terms. Indeed, specifying expectations provides guarantees to customers and helps them to focus on what is the most important.

The most optimal way to specify an SLA is to use a Domain-Specific Language (DSL), it is generally a small programming language dedicated to a specific application domain. It is designed to provide solutions to the problems raised in a particular field and is, therefore, difficult to reuse in other fields. This is why this type of programming language differs from General-Purpose Language (GPL) such as Java, C and C++, whose purpose is to offer solutions to any problem, which may not be the most optimal way, then, the DSL concepts and notations must be close to the objects handled in the treated field [6]. DSLs are created through a process that includes the following steps: decision, analysis, design, implementation, evaluation, and deployment. The decision and analysis determine the need for a DSL as well as the domain model. The design that results from the formalisation of the DSL abstract syntax is then presented. The implementation includes the integration of DSL artefacts into the infrastructure. The evaluation phase is used to validate the DSL, followed by the deployment phase used to deliver the DSL [7].

IoT applications become more complex since several services have to be provided simultaneously with high expected QoS. Indeed, the efficient management of these services is an important issue, hence, one of the solutions is to use a microservices-based approach. Microservices [8] refer to both an architecture and a software development approach which consists in dividing applications into simplest and independent elements. Unlike a classic monolithic approach, where all components form an inseparable entity, microservices aim to divide the business behaviour into small services, which can be run independently of each other. It offers reusability, scalability, extensibility, ease of integration, low coupled architecture, flexibility,

continuous deployment, complexity management and ease of QoS management.

Existing SLA specification languages for IoT applications have limitations such as (a) efficient application management through consideration of the multi-layered nature of IoT, (b) clear and expressive specification of terms and obligations, and (c) consideration of user preferences. However, an SLA specification language for IoT applications must: (1) be domain-specific to handle all IoT constraints (such as the multiplicity of the provided services, the variety of the used devices, and the management of preferences), (2) allow a low coupled architecture to facilitate reusability, scalability, and integration, to ensure the QoS across all layers of the application, and (3) be user preference-aware. Therefore, this paper aims to propose ML-SLA-IoT, a DSL for SLA specification, intended for IoT applications. It differs from other SLA specification languages in the description of user preferences, microservices, and ML-SLA concepts. ML-SLA-IoT allows to:

- Overcome the problems of traditional SLAs in terms of fixed QoS and pricing.
- Facilitate the SLA management, and select the best service offer from the end user's point of view.
- Identify the different resources used in the IoT application.
- Clarify the offered services by fragmenting each of them into a set of microservices for ease of integration, scalability, extensibility, and low coupled architecture.
- Define the obligations of each party and the associated sanctions.

We propose the creation of ML-SLA-IoT, and not to extend one of the existing languages because of: (1) the existing SLA specification languages for IoT are difficult to use because of their complex syntax (e.g. IoT-SLA is specified in JSON, and WSN-SLA is specified in XML), (2) the existing languages do not allow to cover all the essential aspects of a contract for an IoT application (e.g. the consideration of the multi-layered nature of IoT applications), and (3) existing languages are intended for other areas, such as cloud computing, and reorienting the domain of a language causes loss of efficiency and expressiveness. This is why we propose ML-SLA-IoT, which is easy to understand and use, because its syntax is close to human language, and it is practical for IoT application management (because of the use of different concepts such as ML-SLA and microservices). In terms of real world applications, ML-SLA-IoT can be used for the efficient QoS management of the services provided as part of an IoT application, and especially in the phases: (1) contract term negotiation (ML-SLA-IoT provides a high level of clarity and expressiveness), and (2) verification of contract compliance (by monitoring the parameters specified in the SLA).

The remainder of this paper is structured as follows: Section 2 discusses some related work. In Section 3, the proposed SLA specification language is described. Section 4 illustrates the proposed language through a case study. Section 5 presents a comparative study using relevant criteria. In Section 6, the paper is concluded and future work are presented.

## **2. Related work**

Initially, SLAs were written in natural language. However, from the 2000s, the field saw the creation of SLA specification languages, and this language creation mechanism has slowed

down in recent years. In this section, we will present the most used languages. Each area of computer science has its own SLA specification languages, in this section, we will review the best-known languages in each area.

Regarding Cloud Computing, CSLA [9] is proposed to address problems raised by current SLA solutions that are unable to offer an SLA language capable of managing the Cloud dynamic nature, multiple QoS settings, WAN access and fluctuation in end users. A contract in CSLA is made up of parties and obligations. However, CSLA does not consider multi-party contracts, it does not use ML-SLA and does not specify the devices used to provide services.

SLAC [10] is designed to specify SLAs for Cloud Computing. The main differences from the existing specification languages are that SLAC is domain-specific, its semantics are formally defined to avoid ambiguity, and it supports leading Cloud deployment models, and allows the specification of multiparty agreements. An SLA in SLAC consists of: the contract description, the specification of the contract conditions and the guarantee definitions for these conditions. Nevertheless, SLAC does not use ML-SLA and does not specify the devices and user preferences.

In the field of web services, WSLA [11] is a framework for defining and monitoring SLAs for web services. It provides a formal language, an XML schema, and a runtime architecture that interprets the language and manages SLA tasks. It describes the expected service levels about business and technical objectives. The same service can be offered at different levels. An SLA in WSLA describes all the contracted aspects between the signatory parties concerning a specific service, and it consists of parties, service description and obligations. Even though, WSLA is difficult to use because of its complex grammar, it does not consider multi-party contracts, and it does not specify the devices and user preferences.

WS-Agreement [12] is an SLA specification language for Web services. It is an XML-based Web service language and protocol to generate a proposed offer agreement that the agreement initiator wants to establish, negotiate this agreement according to specific constraints, create an agreement between the service provider and the customer with all the conditions and restrictions, and then observe the final compliance. An SLA in WS-Agreement is made up of context, service terms and guarantee terms. Nevertheless, WS-Agreement does not specify the QoS parameters, devices and user preferences, and does not consider multi-party contracts.

For IT services, SLA\* [13] is an SLA specification language, it is a domain-independent syntax. SLA\* was developed as a refinement of XML standards specific to Web services: WS-Agreement and WSLA. Now, instead of Web services, SLA\* is domain-independent, and instead of XML, it is language independent. In SLA\*, a contract consists of the SLA template attributes, the agreement parties, the service descriptions, the declarations of variables, and the agreement terms. Nonetheless, SLA\* does not consider multi-party contracts, and it is not easy to use.

SLALOM [14] is a domain-specific language to reduce the gap between the customer's point of view (business-oriented) and that of the provider (business-oriented towards implementation). SLALOM facilitates SLA specification and monitoring. The authors note that the evaluation of the isolated components does not measure QoS, they, therefore, use SLALOM to support the composition of the measurements from various data sources to present and justify the measurements of these compositions. An SLA in SLALOM is made up of parties and a set of clauses, each addressing an IT service. However, SLALOM does not consider multi-party contracts, and it does not specify the devices and user preferences.

For IoT, WSN-SLA [15] is a model for specifying SLA. The authors extend the WSLA model by integrating elements that meet the constraints of sensor networks. Before the SLA negotiation, the customer must provide the number, position and characteristics list of each device. This device specification describes the sensor subsets with different QoS requirements. An SLA consists of parties, devices, service definitions and obligations. Nevertheless, WSN-SLA does not use ML-SLA and does not specify the QoS parameters and user preferences.

SLA-IoT [3] is an end-to-end SLA specification language for IoT, it is a new conceptual model that captures the knowledge base of IoT-specific SLAs, expressing IoT system entities and their relationships in the SLA context. It proposes a new multilayer grammar to specify SLA for IoT applications. It is a tool for final and standardized SLA specification and composition with a full vocabulary, which provides a finer specification level of the user's needs. A contract is made up of parties, SLOs, workflow activities, services and infrastructure resources. However, SLA-IoT does not specify the penalties, devices and user preferences.

### **3. ML-SLA-IoT: an SLA specification language for IoT systems**

In what follows, we will present ML-SLA-IoT according to the DSL development lifecycle, the first subsection will summarize the first three steps of the lifecycle (decision, analysis and design), and the second subsection will present the last three steps (implementation, evaluation and deployment).

#### **3.1. ML-SLA-IoT meta-model**

The language meta-model has already been presented in [16], in this paper we will present an extended version of the meta-model. In this extension, the third-party has been eliminated and its role has been replaced by the implementation of a monitoring module (outside the scope of this paper). The elimination of the third-party benefits both parties; for the service provider, eliminating the third-party reduces costs and minimises the unnecessary allocation of resources; for the service consumer, the elimination helps ensure data trust and save time due to the automation of parameters control and monitoring. As shown in Figure 1, ML-SLA-IoT is composed of six parts: The first one is responsible for identifying the involved parties in the SLA, ML-SLA-IoT dispenses with the third-party who is responsible for monitoring the SLA terms. The second one is for the identification of the resources used in the IoT application. The third one dedicated to the specification of user preferences, it allows a consumer to determine its preferences in terms of resources and metrics. The fourth one is for the specification of the set of metrics relating to resources. The fifth one is to specify the obligations of each party, these obligations are of two types: guarantees and penalties, a guarantee can be an SLO or a Rule that represents a composition of SLOs. Finally, the sixth one of the language is for specifying the provided services, where each service is divided into a set of microservices for ease of integration, reusability, scalability, extensibility and low coupled architecture (for example, the automatic lighting service is made up of two microservices: the first for lamp management, and the second for automation and control).

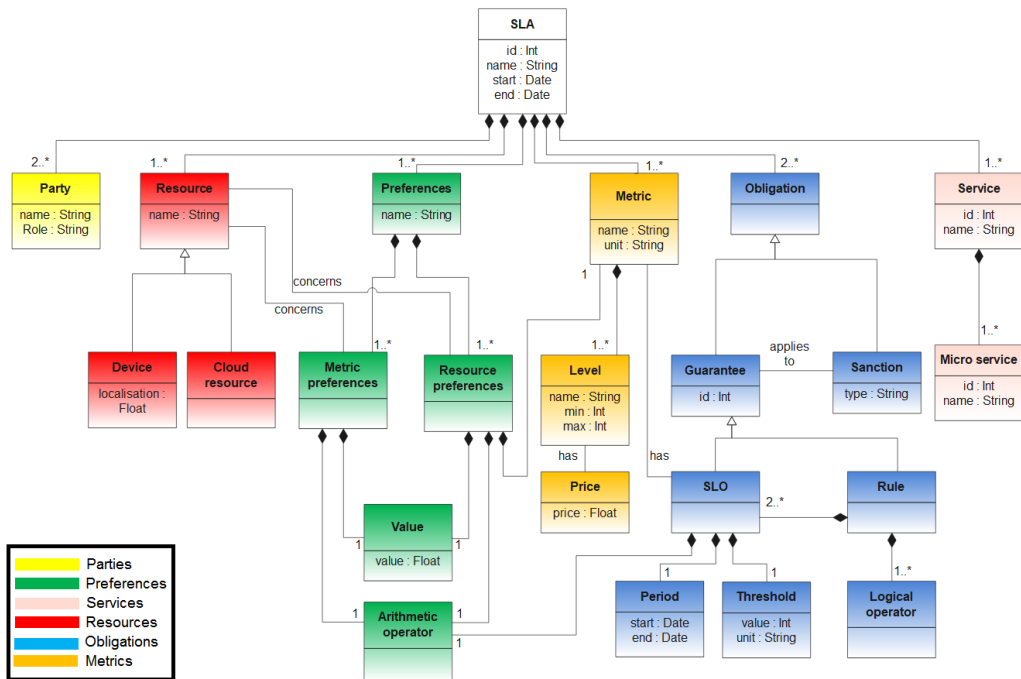


Figure 1: ML-SLA-IoT meta-model

We used the concept of ML-SLA as a solution to:

- Cover all layers of an IoT application (sensing, network, service and interface [17]). As shown in figure 2, metric levels concern the sensing layer, resource specification applies to the network layer, microservices are used for the service layer, user preferences and obligations concern the interface layer.
- Manage the flexibility resulting from the static nature of traditional SLA approaches in terms of QoS and pricing. As shown in figure 3, each QoS metric is divided into three levels where each level has its price.

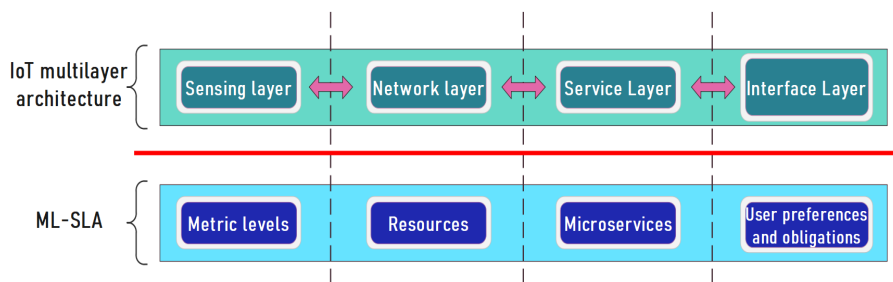
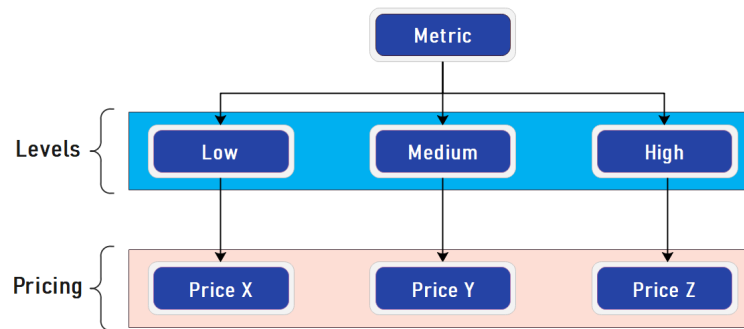


Figure 2: ML-SLA covering all IoT layers



**Figure 3:** ML-SLA for multiple QoS and pricing

### 3.2. ML-SLA-IoT implementation

To implement our language, we used Xtext, which is a free framework developed in Eclipse; Xtext facilitates the implementation of textual DSLs, it works on a JVM (Java Virtual Machine), and it is made up of several APIs that describe the different aspects and offer a full implementation of this language. We use Xtext because it covers all aspects of a complete language infrastructure, starting from the parser, code generator, or interpreter, up to a complete Eclipse IDE integration (with all the typical IDE features). Besides, Xtext generates a grammar description, a parser and an abstract syntax tree. It also offers an initial grammar that can be extended to define the final one. The initial grammar offers important elements such as space management, carriage return, tabulation, strings, integers, etc.

Xtext allows to generate the initial grammar from the meta-model, then, we enrich the grammar by adding the desired rules. Listing 1 shows a part of the ML-SLA-IoT Xtext file, and the Listing 2 shows the rule concerning the specification of a device.

```

1  sla returns SLA:
2  'SLA' Name = EString
3  'Start_date' Start = EString
4  'End_date' End = EString
5  'Parties{'
6      party += Party ( ',' party += Party)*
7  '}'
8  'Consumer_preferences{'
9      preference += Preference ( ',' preference += Preference)*
10 '}'
11 'Services{'
12     service += service ( ',' service += service)*
13 '}'
14 'Micro_Services{'
15     Micro_service += Micro_service ( ',' Micro_service += Micro_service)*
16 '}'
17 'Obligations{'
18     obligation += Obligation ( ',' obligation += Obligation)*
19 '}'
20 'Resources{'
21     resource += Resource ( ',' resource += Resource )*
22 '}'
23 'Metrics{'

```

```

24     metrics += Metric ( ' , ' metrics += Metric)*
25     '}'
26 'end'
27 ;

```

Listing 1: A part of the Xtext file

```

1  Device :
2  'Device { ' name= EString ' , Localisation : ' longitude = EFloat ' , ' latitude =
EFloat ' , ' height = EFloat ' } ' ;

```

Listing 2: The device specification rule

Now, after defining the grammar, we create the Plug-in using Eclipse, which allows us to write an SLA in the proposed language. The proposed SLA is machine-readable to automate monitoring whether the parties hold the agreements and guarantees. On the other hand, it is human-readable to facilitate configuration and maintenance tasks.

## 4. ML-SLA-IoT tool for IoT application

This section illustrates how to create an SLA for an IoT application, using ML-SLA-IoT. We developed a Java application with a useful GUI that facilitate the SLA establishment by enabling, on the one hand, users to enter information related to their preferences in the application form, and on the other hand, the provider to choose options related to the microservices that can be provided in the application. Indeed, it allows the generation of SLA terms according to our proposed microservice and ML-SLA concepts.

As shown in figure 4, the interface contains five tabs: the tab (a) "General information" is used to specify the SLA name, the start and end dates, and the involved parties, the tab (b) "Resources" lists the resources used in the IoT application and their metrics, the tab (c) "Services" indicates the microservices that must be provided, the tab (d) "Preferences" shows user preferences which are resource and metric preferences, and the tab (e) "Obligations" allows to express SLOs and rules.

To show the usability and efficiency of ML-SLA-IoT, we have used a reduced example of Smart Home, in which the consumer wants a service for the control of the ambient temperature, the consumer wants to have three air conditioners, and that the temperature is below 23 °C in summer, and a short response time. This SLA is made up of a single service which is the ambient temperature control. This service comprises two microservices: the air conditioner adjustment and temperature monitoring. The resources used are temperature sensors and air conditioners. The metrics associated with these devices is temperature and response time. The SLA has two SLOs and a Rule, the first SLO concerns ambient temperature, and the second concerns device response time.

The SLA will be automatically generated after filling in all the fields of the tabs and confirming, as shown in Listing 3, and the elements listed below are the key points of the language:

- **Line 26 to 30:** ML-SLA-IoT focuses on the consumer by specifying his preferences to precisely target the desired goals;



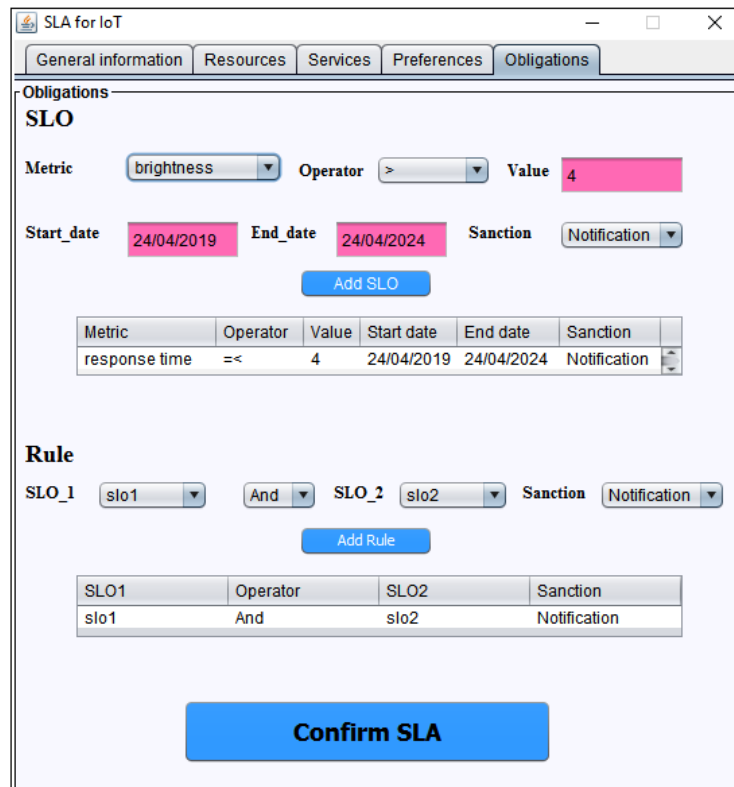


Figure 4: ML-SLA-IoT graphical interface

- **Line 12 to 15:** the use of microservices allows efficient management due to weak coupling according to two points: the first one is flexibility in the case of dynamic requirement changes, and the second one is the resource optimization;
- **Line 36:** rules allow the combination of SLOs to facilitate the management of obligations;
- **Line 22 and 23:** the use of multi-level metrics to dispense with problems of fixed pricing and QoS level change.

```

1  SLA "SLA_for_Smart_Home"
2  Start_date "03/03/2021"
3  End_date "03/03/2022"
4  Parties {
5      Customer { "William_Smith" },
6      Provider { "Bosch_Smart_Home" }
7  }
8
9  Services {
10     Service [ "Ambient_temperature_control", list_of_microservices {
11         Temperature_monitoring, Air_conditioner_adjustment } ]
12 }

```

```

13     micro_service( "Temperature_monitoring", id1, list_of_resources[
Temperature_sensor])
14     micro_service( "Air_conditioner_adjustment", id1, list_of_resources[
Air_conditioner])
15 }
16
17 Resources{
18     Device[ "Temperature_sensor", localisation(16.2,23.3,14.0)]
19     ,Device[ "Air_conditioner", localisation(4.2,6.3,1.0)]
20 }
21 Metrics{
22     metric[ "Temperature" , level "High" ],
23     metric[ "Response_time" , level "Medium" ]
24 }
25
26 Consumer_preferences{
27     metric_preference[Temperature in Air_conditioner < 23]
28     ,metric_preference[Response_time in Air_conditioner < 3]
29     ,resource_preference[Air_conditioner = 3]
30 }
31
32 Obligations{
33     Sanction[1, "Notification" ], Sanction[2, "Compensation" ],
34     SLO[id: "SLO1" Response_time > 3 for_period [ "03/03/2020" to "03/03/2021" ]
sanction = Compensation ],
35     SLO[id: "SLO2" Temperature < 23 for_period [ "03/03/2020" to "03/03/2021" ]
sanction = Notification ],
36     Rule[SLO1 And SLO2 sanction = Compensation ]
37 }
38
39 end

```

Listing 3: SLA for a Smart home specified in ML-SLA-IoT

## 5. Discussion

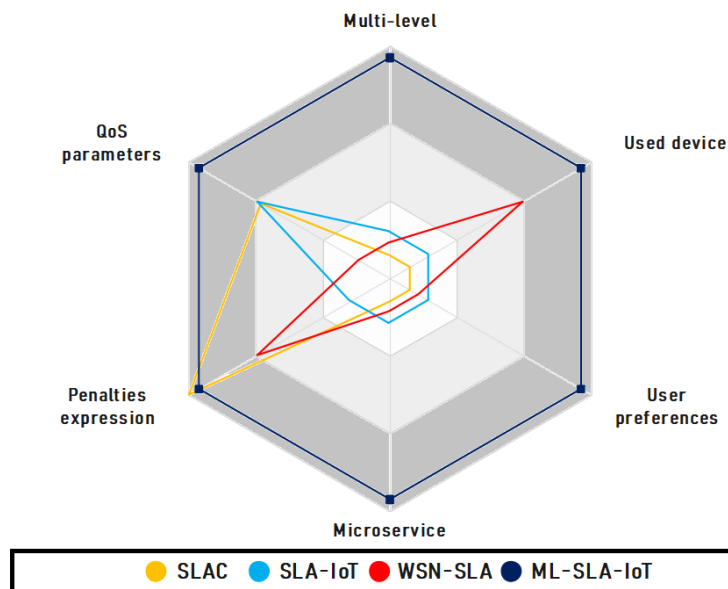
Current SLA specification languages have limitations, such as the expression of user preferences, fixed pricing despite QoS degradation, poor service specification, and tightly coupled service architecture that affects management, scalability, and service provision. For that, we proposed ML-SLA-IoT, which is an SLA language specification intended for IoT, it allows to: identify the involved parties, express the consumer preferences, identify the different resources used in the system, clarify the offered services by fragmenting each of them into a set of microservices, use ML-SLA in the partition of metrics in several levels to optimise the management and the provision of services, define the obligations of each party and the associated sanctions. Figure 5 presents a comparison diagram between ML-SLA-IoT (our proposed language) and some languages close to our contribution, this diagram includes two levels:

- **Unsupported criterion:** (the white zone) means that the language does not support the criterion.
- **Supported criterion:** means that the solution supports the criterion, it includes two sub-levels:
  - *Low:* (the light gray zone) this evaluation indicates that the criterion is partially covered.

- *High*: (the dark gray zone) this evaluation indicates that the criterion is completely covered.

We discussed and examined the languages based on the following criteria:

- **Multi-level**: indicates if the language takes into account the notion of ML-SLA.
- **QoS parameters**: show if the solution specifies the QoS parameters or not.
- **Microservice**: expresses whether the notion of microservices is considered.
- **Penalties expression**: indicates whether the language expresses penalties. If it is the case, it covers all the SLA life cycle, otherwise, it does not cover it.
- **User preferences**: indicates whether the language takes into account user preferences.
- **Used devices**: indicates if the language defines the different devices used in the service provision.



**Figure 5:** Comparison of SLA specification languages

As shown in figure 5, no language allows to specify the user preferences, use the microservice approach to specify services, and support the ML-SLA concept. Only ML-SLA-IoT and WSN-SLA allow specifying the used devices.

## 6. Conclusion and future work

This paper has presented ML-SLA-IoT, an SLA specification language for IoT applications, it is a clear, simple, expressive and well-described language, it is user preference-aware allowing to ensure that the provided QoS meets the agreed consumer expectations, it plays a central role in selecting the best service offer from the end user's point of view, and it covers customer's

requirements to facilitate the SLA management. ML-SLA-IoT is composed of six parts: The first one is responsible for identifying the involved parties in the SLA, such as the consumer and the provider. The second one is dedicated to the specification of user preferences, it allows a consumer to determine its preferences in terms of resources and metrics. The third one is for specifying the provided services, where each service is divided into a set of microservices for ease of integration, reusability, scalability, extensibility and low coupled architecture. The fourth one is for the identification of the resources used in the IoT application. The fifth one is for the specification of the set of metrics relating to resources, for the parameterised QoS level management, we used ML-SLA to solve problems related to the static nature of traditional SLA approaches in terms of QoS and pricing. Finally, the sixth one is to specify the obligations of each party, these obligations are of two types: guarantees and penalties, a guarantee can be an SLO or a Rule that represents a composition of SLOs.

As long as services evolve, their SLAs become more complicated, even if some points have already been explored. As a result, challenges remain in SLA management in terms of scalability, dynamic environmental changes, heterogeneity, etc. Regarding our future work, we intend to integrate a monitoring module that uses ML-SLA-IoT language to provide and monitor the QoS as efficient and customized as possible in smart healthcare systems.

## Acknowledgments

This work was partially supported by the LABEX-TA project MeFoGL: "Méthodes Formelles pour le Génie Logiciel".

## References

- [1] I. U. Haq, A. A. Huqqani, E. Schikuta, Hierarchical aggregation of service level agreements, *Data & knowledge engineering* 70 (2011) 435–447.
- [2] L. Wu, R. Buyya, Service level agreement (SLA) in utility computing systems, in: *Performance and dependability in service computing: Concepts, techniques and research directions*, IGI Global, 2012, pp. 1–25.
- [3] A. Alqahtani, E. Solaiman, R. Buyya, R. Ranjan, End-to-end QoS specification and monitoring in the internet of things, *IEEE Technical Committee on Cybernetics for Cyber-Physical Systems* (2016).
- [4] P. Grubitzsch, I. Braun, H. Fichtl, T. Springer, T. Hara, A. Schill, ML-SLA: Multi-level service level agreements for highly flexible IoT services, in: *2017 IEEE International Congress on Internet of Things (ICIOT)*, IEEE, 2017, pp. 113–120.
- [5] A. Keller, H. Ludwig, The WSLA framework: specifying and monitoring service level agreements for web services, *Journal of Network and Systems Management* 11 (2003) 57–81.
- [6] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler, S. Völkel, Design guidelines for domain specific languages, in: *Proceedings of the 9th OOPSLA Workshop on Domain-Specific Modeling*, Helsinki: Helsinki School of Economics, 2009, pp. 7–13.

- [7] A. Barišić, V. Amaral, M. Goulão, Usability driven DSL development with USE-ME, *Computer Languages, Systems & Structures* 51 (2018) 118–157.
- [8] H. Vural, M. Koyuncu, S. Guney, A systematic literature review on microservices, in: *International Conference on Computational Science and Its Applications*, Springer, 2017, pp. 203–217.
- [9] Y. Kouki, *Service Level Agreement-Driven Approach to Cloud Elasticity Management*, Ph.D. thesis, Nantes, Ecole des Mines, 2013.
- [10] R. B. Uriarte, F. Tiezzi, R. D. Nicola, SLAC: A formal service-level-agreement language for cloud computing, in: *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, IEEE Computer Society, 2014, pp. 419–426.
- [11] H. Ludwig, A. Keller, A. Dan, R. P. King, R. Franck, Web service level agreement (WSLA) language specification, *Ibm corporation* (2003) 815–824.
- [12] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, M. Xu, Web services agreement specification (WS-Agreement), in: *Open grid forum* (2007), volume 128, 2007, p. 216.
- [13] K. T. Kearney, F. Torelli, C. Kotsokalis, SLA\*: An abstract syntax for Service Level Agreements, in: *2010 11th IEEE/ACM International Conference on Grid Computing*, IEEE, 2010, pp. 217–224.
- [14] A. Correia, V. Amaral, et al., SLALOM: a Language for SLA Specification and Monitoring, *arXiv preprint arXiv:1109.6740* (2011).
- [15] G. Gaillard, D. Barthel, F. Theoleyre, F. Valois, SLA Specification for IoT Operation - The WSN-SLA Framework, *Technical Report 8567*, INRIA, 2014. URL: <http://icube-publis.unistra.fr/7-GBTV14>, 71 pages.
- [16] N. Staifi, M. Belguidoum, Adapted smart home services based on smart contracts and service level agreements, *Concurrency and Computation: Practice and Experience* (2021) e6208.
- [17] L. Da Xu, W. He, S. Li, Internet of things in industries: A survey, *IEEE Transactions on industrial informatics* 10 (2014) 2233–2243.