AsyncSLA: Towards a Service Level Agreement for Asynchronous Services

Marc Oriol  
Universitat Politècnica de Catalunya  
Barcelona, Spain  
marc.oriol@upc.edu

Abel Gómez  
Universitat Oberta de Catalunya  
Barcelona, Spain  
agomezla@uoc.edu

Jordi Cabot  
Luxembourg Institute of Science and Technology  
Luxembourg, Luxembourg  
jordi.cabot@list.lu

ABSTRACT
Complex distributed systems increasingly involve physical components as part of cyber-physical systems or Internet of things initiatives. Communication with such subsystems is typically asynchronous. Several initiatives like Web of Things or AsyncAPI have emerged to standardize and facilitate the definition of such asynchronous communications. However, these initiatives do not cover standards to specify the quality of service or define service level agreements (SLAs) for these asynchronous interactions. To address this issue, this paper proposes both a comprehensive quality model for asynchronous services based on the ISO/IEC 25010 standard, and a domain specific language to specify SLAs for asynchronous services based on the WS-Agreement standard. To facilitate its adoption, our proposed solution has been expressed also as an extension for the AsyncAPI specification. Finally, we provide a tool support to define these SLAs by extending an existing toolkit.

KEYWORDS
AsyncAPI, Web of Things, Service Level Agreement.

ACM Reference Format:

1 INTRODUCTION
Physical components are becoming more pervasive and are often equipped with sensors and actuators that allow them to interact with their environment and with each other in what is called the Internet of things (IoT). Systems mixing this kind of components are called cyber-physical systems (CPS). CPS are the key element of modern industrial environments, known as Industry 4.0 [21], which are becoming widespread in several sectors, enabling smart cities, smart homes, and smart transportation [17].

These physical components communicate with each other through the set of services they expose. These services are typically asynchronous—e.g., a service may continuously provide context information obtained from the physical sensors of the underlying component. All these asynchronous communications have led to an architectural style known as event-driven service-oriented architecture [11]. In these architectures, the constituent services communicate through the notification of events following the publish-subscribe pattern: a service producer publishes some event into a service broker, which sends it to all the service consumers subscribed to that type of events. In this manner, the interaction between producers and consumers is highly decoupled, which facilitates its scalability, reduces dependencies, and improves fault tolerance [11].

Event-driven service-oriented architectures have surged in popularity because of its adequacy in the context of CPS. According to a 2021 industry survey conducted by Solace, 85% of organizations recognize the business value and are striving for event-driven SOA architectures [28]. However, a subsequent survey conducted by the same organization in 2022 found that the biggest technical challenge for adopting this architecture is its integration with existing applications [29].

Several initiatives have emerged to facilitate the integration of this type of IoT services. For instance, the Web of Things (WoT) proposal aims to reuse and extend standard web technologies to apply them to the IoT domain [26]. In the same direction, specifications like AsyncAPI—which is independent of the underlying technology and protocol, and as such, can describe services using web-based technologies like WebSocket—have emerged to provide machine-readable definitions of APIs of asynchronous services [5].

Although these initiatives provide a common framework that enables the definition, discoverability, and interoperability of IoT-based services, they do not provide standards to define the quality of service (QoS) that is expected or required from them. For example, latency, availability, or throughput are key QoS metrics that need to be guaranteed in order to ensure a proper operation of the system. Although several approaches have been proposed to define and specify the QoS in the fields of WoT and IoT, they are limited in the number and type of metrics they address. Moreover, the required QoS is usually formalized in the form of a service level agreement (SLA). SLAs are contracts that establish the QoS agreed between service providers and service consumers [24]. Several formal machine-readable languages have been proposed for specifying SLAs. The primary objective behind making them machine-readable is so that they can be processed by a computer and, for instance, automatically derive and instantiate monitors to measure the compliance of the conditions established in the SLA. Although some proposals for
SLAs for WoT and IoT exists, they have some limitations in their expressivity and lack alignment with current standards, such as WS-Agreement or the AsyncAPI specification (see Section 2 for an in-depth discussion).

To cover this gap, this paper proposes (i) a quality model that comprises and structures several quality dimensions for asynchronous services based on the ISO/IEC 25010 standard; (ii) a domain specific language (DSL) to specify SLAs for asynchronous services considering these dimensions, adapting the WS-Agreement standard and integrating it with the AsyncAPI specification; and (iii) a tool that supports the proposed language and quality model.

We argue that these contributions are a first step towards supporting automatic techniques that rely on machine-readable QoS in an IoT or asynchronous context, such as the automatic generation of monitors, SLA assessment, or the discoverability of WoT based on QoS.

The rest of the paper is organized as follows: Section 2 discusses the related work. Section 3 explores the quality of service for asynchronous services and introduces our proposed quality model. Section 4 details our proposed domain specific language to specify SLAs for asynchronous services. Section 5 describes the tool support we have developed. Finally, Section 6 concludes the paper.

2 RELATED WORK

Several languages have been proposed in the literature to specify SLAs for traditional synchronous web services. Some of the most well known are WSLA [16], SLAng [18], SLA* [15] and WS-Agreement [4], with the latter being the established standard endorsed by the Open Grid Forum. In comparison, SLA languages for IoT-based asynchronous services is rather limited. We have analyzed some of the most relevant SLA proposals in the literature for asynchronous services along with the characteristics of the most well known synchronous ones. In this analysis we have examined various general and technical characteristics of the SLA languages. The general characteristics that we have examined comprise the type of service, metrics supported, the level of the defined syntax model, and whether the proposal is derived from a standard. The technical characteristics that we have examined comprise if the SLA supports preconditions, complex conditions, different scopes and whether the proposal has tool support. The Table 1 provides a summary of such analysis that we describe in detail below.

Type of service: The proposals analysed cover both synchronous and asynchronous services. Regarding the synchronous ones, some SLA languages are aligned to a particular web service specification: SLA4OAI [9] is aligned with the OpenAPI specification used in RESTful services, and WSLA [16] and SLAng [18] are aligned with the Web Service Description Language (WSDL), which is usually used for SOAP-based web services. Other proposals are generic and are not aligned with any particular language describing the web service specification [4, 15]. However, due to this generalizability, they do not provide a fully concrete syntax model. Regarding the asynchronous services, none of the proposals analysed are aligned with the AsyncAPI specification.

Metrics: Most of the proposals—both synchronous and asynchronous—offer support for an extensible set of custom metrics [3, 4, 9, 15, 16, 20]. However, it is worth noting that these metrics are often presented as illustrative examples without a standardized structure or clear definitions. Furthermore, a few proposals for asynchronous services are limited to just a few metrics related to availability and latency without providing extensibility capabilities [8, 23]. It is noteworthy that only SLAng [18] introduces a quality model that systematically organizes the metrics employed in the SLA. However, it is important to note that the quality model presented in SLAng has been defined without considering any standard as a reference.

Syntax model: Some proposals provide SLAs as simple examples without any formalism [8, 23]. Some of the proposals provide only an abstract representation, such those presenting a UML meta-model [20], or an abstract set of mathematical rules [15] without a concrete syntax specifying the grammar of the SLA. Only a few proposals provide a complete DSL consisting of an abstract and a concrete syntax for the language [3, 9, 16, 18]. And from these, only Alqahtani et al. [3] has been designed for asynchronous services. It is also important to remark that WS-Agreement [4] provides a concrete syntax, but only partially, as many elements have been not defined on purpose in favour of generalizability, leading to multiple WS-Agreement complaint (sub)languages.

Derived from a standard: Several of the proposals of SLA languages for synchronous services have become or are based on some standard: WSLA, developed by IBM, was one of the first languages considered a de-facto standard due to its popularity in the early 2000s. This was the case, until the irruption of WS-Agreement, which was endorsed by the OSGi group, and emerged as the new standard for synchronous services. Some of the SLA language proposals for synchronous services have been derived from one or both of these two languages [9, 15]. However, regarding asynchronous services, almost none of the proposals derive from a standard (including the standards established in the synchronous services domain). Only Li et al. [20] is based on the WS-Agreement standard and has been adapted for asynchronous services.

Support for preconditions: Some SLA languages for synchronous services provide support to define preconditions in their guarantee terms (i.e. a precondition that must be met for a guarantee to be enforced), either completely [4, 15] or partially (e.g. limited to the periods of time when the guarantee should be enforced) [16]. However, in the context of asynchronous services, only WoT-SLA [20] provides such support.

Support for complex conditions: Some proposals for synchronous services support the definition of complex boolean expressions in the objectives to be met (e.g. a combination of nested AND/OR operators) [4, 15, 16]. However, none of the SLAs for asynchronous services provide this type of support, limiting the expressivity of the guarantees to simple conditions consisting of a metric, an operand and a value.

Support for different scopes: An objective may apply to a specific scope of a service, such as a service method in synchronous services or a channel in asynchronous ones. Although a few proposals do not support the definition of scopes in their SLA languages, most of them do [3, 4, 9, 15, 16, 20].

Tool support: With the exception of Mustafa et al. [23], all proposals have tool support for their proposed SLA language.

To conclude, although some solutions exists defining SLA languages for asynchronous services, their capabilities are not as powerful as those proposed for synchronous ones. To cover this gap,
Table 1: Analysis of the related work

<table>
<thead>
<tr>
<th>Proposal</th>
<th>Type of service</th>
<th>Metrics</th>
<th>Syntax model</th>
<th>Derived from standard</th>
<th>Support for preconditions</th>
<th>Support for complex conditions</th>
<th>Support for different scopes</th>
<th>Tool support</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSLA [16]</td>
<td>Synchronous (WSDL)</td>
<td>Extensible, custom</td>
<td>Concrete</td>
<td>Yes*</td>
<td>Partially</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SLAng [18]</td>
<td>Synchronous (WSDL)</td>
<td>Extensible, based on quality model</td>
<td>Concrete</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SLA* [15]</td>
<td>Synchronous (generic)</td>
<td>Extensible, custom</td>
<td>Abstract</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>WS-Agreement [4]</td>
<td>Synchronous (generic)</td>
<td>Extensible, custom</td>
<td>Concrete (Partially)</td>
<td>Yes*</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SLA4OAI [9]</td>
<td>Synchronous (OpenAPI)</td>
<td>Extensible, custom</td>
<td>Concrete</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>WIoT-SLA [20]</td>
<td>Asynchronous</td>
<td>Extensible, custom</td>
<td>Abstract</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Ezzedine et al. [8]</td>
<td>Asynchronous</td>
<td>Limited, custom</td>
<td>Not formalized</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Alqahtani et al. [3]</td>
<td>Asynchronous</td>
<td>Extensible, custom</td>
<td>Concrete</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Mustafa et al. [23]</td>
<td>Asynchronous</td>
<td>Limited, custom</td>
<td>Not formalized</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>AsyncSLA</td>
<td>Asynchronous (AsyncAPI)</td>
<td>Extensible, based on quality model</td>
<td>Concrete</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

we propose an SLA language for asynchronous services aligned with the AsyncAPI specification and comprising an extensible set of metrics based on a quality model; with an abstract and concrete syntax model, derived from the WS-Agreement standard, including support for preconditions, complex conditions, multiple scopes, and tool support.

3 QUALITY OF SERVICE FOR ASYNCHRONOUS SERVICES

QoS is defined by the ISO/IEC 13236:1998 standard as “A set of qualities related to the provision of a service as perceived by a service-user” [13]. These qualities comprise many dimensions of the service—e.g., performance, security, reliability, etc. In this regard, the first question when proposing a QoS approach for asynchronous systems is: what are the quality dimensions we should be concerned about and what metrics can be used to measure them? For instance, time performance may be evaluated with the response time in synchronous services (i.e., the time it takes for the service client to receive the response since the request was sent). However, in the field of asynchronous services, the asynchronous interaction of the services requires other metrics to measure performance (e.g., the time it takes from the packet generation by the source node to the packet reception by the destination node).

To address this question, quality models have been proposed as the engineering artifacts to structure the concepts and definitions of the quality dimensions of a software system [12]. There exist many proposals of general-purpose quality models for software systems. They differ on the terminology that they use, the set of quality attributes they define, and the structure of the quality model. The most widely adopted one is the ISO/IEC 25010 standard [12], which classifies software quality into a structured set of high-level characteristics—functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability—which are then decomposed into 31 sub-characteristics (see Figure 1). However, the ISO/IEC 25010 standard is not specifically tailored to the services domain. For instance, the sub-characteristic Installability is usually not applicable to web services because they are executed remotely at the server side. On the other hand, the ISO/IEC 25010 standard does not provide quantitative or qualitative measurable metrics to evaluate those dimensions.
While many quality models tailored for web services have been proposed [25], to the best of our knowledge, no comprehensive quality model specific for asynchronous services has been defined yet. As a result, most works involving QoS for asynchronous services usually use quality dimensions without a formal structure, as discussed in the related work.

To cover this gap, we propose a quality model for asynchronous services. To build this quality model, we gathered a variety of existing quality metrics and their definitions as proposed in the literature, both from services in general (i.e. they apply to both synchronous and asynchronous services) and asynchronous services in particular. Instead of building this quality model from scratch, we extended the quality model derived from a systematic study of quality models for services, which was developed as part of one of the authors’ previous work [25]. This earlier study identified several quality metrics for services and categorized them to the sub-characteristics of the ISO/IEC 25010 standard.

In this work, we have identified the quality metrics that are applicable to asynchronous services from the quality model presented in [25] and, when needed, updated their definitions according to the results we found in the literature for asynchronous services, resulting in a quality model for asynchronous services aligned with the ISO/IEC 25010 standard.

For each metric, we provide its definition and a reference to the work that inspired it—if not, it is based on our expertise in the topic. We also group these metrics based on the most appropriate quality sub-characteristic from ISO/IEC 25010. Table 2 shows the core metrics we identified, mapped to its corresponding sub-characteristics of the ISO/IEC 25010 standard.

It is worth to mention, that the presented metrics are independent from the monitoring process, although their measurements might be affected by the monitoring infrastructure in place. For instance, latency is defined as the time it takes from the packet generation by the source node to the packet reception by the destination node. If the monitors are instrumented in both the service producer and service consumer, the latency measured is end-to-end. However, the monitors might be instrumented between the service producer and the service broker, measuring therefore the latency between these two entities. It is also worth noting that other type of metrics do not require to instrument both sides of the communication and can be measured by instrumenting just one side (e.g. Payload size). Finally, some metrics are rather static and do not require runtime monitoring (e.g. versioning).

4 A DOMAIN SPECIFIC LANGUAGE FOR SLAS OF ASYNCHRONOUS SERVICES

We propose a domain specific language (DSL) to define SLAs for asynchronous services, named AsyncSLA, by tailoring the structure and key concepts of the WS-Agreement standard, as this will help in the adoption and recognition of the SLA to all people familiar with this specification.

In addition to addressing the quality dimensions for asynchronous services outlined in the previous section, we have also specified our DSL to support the inherent characteristics of asynchronous services, that are distinct from the synchronous ones. These characteristics are:

- **Interactions are initiated by the service provider**: In contrast to synchronous services where the service client initiates the interaction—i.e., the service client sends a request to the service provider to obtain a response—in asynchronous services, it is the service producer who initiates the interaction—i.e., the service producer sends the information related to an event to the subscribed service consumers.
- **One-way interaction with service consumers**: Although the service consumer can acknowledge the reception of the information, there is no response that the service producer expects from its subscribed consumers.
- **Channel-based communication**: Instead of service methods or operations that are offered by synchronous services, asynchronous services are composed of channels, where the service producer publishes the information related to an event...
<table>
<thead>
<tr>
<th>Quality Metric</th>
<th>Parent from ISO/IEC 25010</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Probability of correctness</td>
<td>Functional suitability (correctness)</td>
<td>The probability that the content of the message accurately represents the corresponding real world situation [1]</td>
</tr>
<tr>
<td>Precision</td>
<td>Functional suitability (correctness)</td>
<td>How exactly the provided information in the message mirrors the reality. Precision is specified with bounds [1].</td>
</tr>
<tr>
<td>Up-to-dateness</td>
<td>Functional suitability (correctness)</td>
<td>How old is the information upon its receipt [1]</td>
</tr>
<tr>
<td>Latency</td>
<td>Performance (time behaviour)</td>
<td>The total time from the packet generation by the source node to the packet reception by the destination node [6]</td>
</tr>
<tr>
<td>Round Trip Time</td>
<td>Performance (time behaviour)</td>
<td>The time from the packet generation by the source node to the reception of the ACK [27]</td>
</tr>
<tr>
<td>Jitter</td>
<td>Performance (time behaviour)</td>
<td>Variation of Latency [1]</td>
</tr>
<tr>
<td>Throughput</td>
<td>Performance (time behaviour)</td>
<td>The amount of error-free bits transmitted in a given period of time [19]</td>
</tr>
<tr>
<td>Message frequency</td>
<td>Performance (time behaviour)</td>
<td>The amount of messages transmitted in a given period of time. It is the inverse of Time period</td>
</tr>
<tr>
<td>Time period</td>
<td>Performance (time behaviour)</td>
<td>The amount of time elapsed between two messages. It is the inverse of message frequency</td>
</tr>
<tr>
<td>Memory used</td>
<td>Performance (resource utilization)</td>
<td>The amount of memory used by the service</td>
</tr>
<tr>
<td>CPU used</td>
<td>Performance (resource utilization)</td>
<td>The amount of CPU used by the service</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>Performance (resource utilization)</td>
<td>The amount of bandwidth used by the service</td>
</tr>
<tr>
<td>Payload size</td>
<td>Performance (resource utilization)</td>
<td>The size of the payload enclosed in the message sent by the service</td>
</tr>
<tr>
<td>Max. subscribers</td>
<td>Performance (capacity)</td>
<td>The maximum amount of subscribers the service is capable of handling</td>
</tr>
<tr>
<td>Max. throughput</td>
<td>Performance (capacity)</td>
<td>The maximum throughput the service is capable of providing</td>
</tr>
<tr>
<td>Load balancing</td>
<td>Performance (capacity)</td>
<td>To what extent the service has load balancing capabilities</td>
</tr>
<tr>
<td>CPU capacity</td>
<td>Performance (capacity)</td>
<td>The maximum CPU the service is capable of using</td>
</tr>
<tr>
<td>Memory Capacity</td>
<td>Performance (capacity)</td>
<td>The maximum memory the service is capable of using</td>
</tr>
<tr>
<td>Documentation</td>
<td>Usability (recognizability)</td>
<td>The extent to which a web service interface contains an adequate level of documentation [2]</td>
</tr>
<tr>
<td>Discoverability</td>
<td>Usability (recognizability)</td>
<td>The ease and accuracy of consumers’ search for an individual service, based on the description of the service [14]</td>
</tr>
<tr>
<td>Expected failures</td>
<td>Reliability (maturity)</td>
<td>Expected number of failures over a time interval [7]</td>
</tr>
<tr>
<td>Type Consistency</td>
<td>Reliability (maturity)</td>
<td>The data flow is consistent with respect to definition of data types [1]</td>
</tr>
<tr>
<td>Time to Fail</td>
<td>Reliability (maturity)</td>
<td>Time the service runs before failing [7]</td>
</tr>
<tr>
<td>Availability</td>
<td>Reliability (availability)</td>
<td>The probability that a service is available at any given time [1]</td>
</tr>
<tr>
<td>Uptime</td>
<td>Reliability (availability)</td>
<td>The duration for which the service has been operational continuously without failure [22]</td>
</tr>
<tr>
<td>Packet loss</td>
<td>Reliability (availability)</td>
<td>Ratio of the number of undelivered packets to sent ones [7]</td>
</tr>
<tr>
<td>Disaster resilience</td>
<td>Reliability (fault tolerance)</td>
<td>How well the service resists natural and human-made disasters [22]</td>
</tr>
<tr>
<td>Exception Handling</td>
<td>Reliability (fault tolerance)</td>
<td>Internal activities that are performed in the case of failures during the execution of the service. [1]</td>
</tr>
<tr>
<td>Time to Repair</td>
<td>Reliability (recoverability)</td>
<td>Time needed to repair and restore service after a failure [7]</td>
</tr>
<tr>
<td>Failover</td>
<td>Reliability (recoverability)</td>
<td>Whether the service employs failover resources, and how quickly [22]-</td>
</tr>
<tr>
<td>Versioning</td>
<td>Maintainability (Modifiability)</td>
<td>Whether the service implements versioning strategies to ensure backward compatibility</td>
</tr>
</tbody>
</table>

so that the service consumers subscribed to that channel can receive it. • **Broadcast communication:** When an asynchronous service publishes a message, this is received by all the subscribed service consumers in the channel, leading to a 1–N relationship
for a given event (i.e. one event has one service producer and N service consumers receiving it).

Fig. 2 depicts the metamodel—i.e., the modeling primitives and their relationships—of our DSL for asynchronous services SLAs, following the WS-Agreement structure. It is worth noting that the figure does not depict it as a conceptual model, but as an implementation model using the Ecore metamodeling language of the Eclipse Modeling Framework (EMF) [30]. We present this model for brevity, since we also cover the concrete notation of our DSL in Section 5.

SLA is the top-level class for the definition of an SLA, and it is composed of several GuaranteeTerms. A GuaranteeTerm is a WS-Agreement element that defines the terms that the service must meet regarding the QoS. It includes a set of scopes, a set of qualifying conditions and a set of service level objectives (SLO). While these components are inherent to the WS-Agreement standard, we’ve tailored them to suit asynchronous services. The Scope defines the scope where the GuaranteeTerm applies to. In our DSL, it includes an identifying name, and relates to the service Channel that is used as a scope—which is akin to the service methods in synchronous services. This granularity enables tailored QoS conditions for different service channels within the asynchronous service. For instance, one channel providing information related to critical events may impose stricter QoS conditions compared to another less critical channel. The QualifyingCondition defines the preconditions, expressed as a boolean expression (BooleanExpression), that must be met for a condition to be enforced. In its turn, the SLO defines the QoS condition that must be assessed—when the QualifyingCondition is met. An SLO is specified as a boolean expression of the condition that must be assessed for the goal to be fulfilled.

In our DSL, boolean expressions support the combination of OrExpressions and AndExpressions in a tree-structure form, leading to one or more ComparisonExpressions. A ComparisonExpression is an expression composed of a QoS Metric, an Operator, and a value. Based on our findings in different SLAs, we opted for a lightweight but powerful enough grammar to define boolean expressions, rather than using a more complex but cumbersome language—e.g., OCL.

QoS Metric defines the metric involved in the SLA/SLO evaluation. It has an identifying name, a description, a QoS Metric Type, a QoS Metric Unit, and a groupedByMessage flag. The attribute QoS Metric Type is used to link the defined metric to the type of metrics from the quality model. In this way, the SLA can be seamlessly integrated with any predefined quality model, allowing flexibility in linking the defined metric to specific metric types through the QoS Metric Type attribute. The groupedByMessage flag determines whether the QoS Metric is calculated as the average result for the metric across all clients subscribed to a specific event when set to ‘true,’ or if distinct QoS metrics are calculated for each individual client when set to ‘false.’ For instance, if two clients are subscribed to the same channel and this attribute is set to ‘false,’ two values (one per client) would be produced. Conversely, if it’s set to ‘true,’ a single value (the average of both clients’ metrics) would be generated.

Finally, DerivedQoS Metric defines a derived QoS metric, i.e., the result of aggregating values—e.g., average, max, min—in an interval. Thus, it includes a window, a windowUnit, and an aggregationFunction.

5 A TOOL TO SPECIFY SLAS FOR ASYNCHRONOUS SERVICES

To show the feasibility of our approach, we have implemented a tool prototype that supports the definition of SLAs for asynchronous services using our AsyncSLA proposal.

To implement this tool, we have first defined a concrete syntax matching the metamodel described in the previous section. In particular, the concrete syntax is defined as an extension of the AsyncAPI specification as we believe this will facilitate its adoption. AsyncAPI is an initiative that, similar to what OpenAPI represents for the REST world, aims at providing a single source of truth in the definition of asynchronous and message-based services. AsyncAPI

Listing 1: Excerpt of an example of SLA integrated with the AsyncAPI specification

```json
{
    "asyncapi": "2.0.0",
    "channels": {
        "smartylighting/.../lighting/measured": {}
    }
}
}```
descriptions are expected to be both human and machine readable. To achieve this goal, files defining a message-driven API are represented as JSON objects and conform to the JSON standards. Such files allow describing, among other things, the message brokers of an architecture, the topics—i.e., channels—of interest, or the formats of the messages associated with each of the topics.

As part of our previous work, we developed the AsyncAPI Toolkit\(^1\), an Eclipse-based tool that supports the development of AsyncAPI specifications and automatically generate code using model-driven engineering techniques. The AsyncAPI Toolkit defines both the concrete and abstract syntax for AsyncAPI specifications using Xtext\(^2\).

We have now extended the AsyncAPI Toolkit to seamlessly integrate both AsyncAPI and the AsyncSLA specifications. This integration starts at the metamodel level, where we have linked the AsyncAPI metamodel presented in our previous work [10] with the AsyncSLA metamodel presented in Section 4. In the integration, we have fused the common elements such as Channel, and we have added the SLA object as a field of the root AsyncAPI object.

We have also defined a concrete syntax for the SLA-related elements—i.e., those that do not already belong to the AsyncAPI specification—in JSON. We indicate that such additions do not conform the official AsyncAPI specification by prefixing them with an x-. An example of an SLA using our concrete syntax integrated in the AsyncAPI specification is shown in Listing 1.

This example shows an SLA that defines 3 metrics:

- availabilityPerEvent is a metric of type availability. By setting the flag groupedByEvent to true, it measures to what extent the service provider is available to all subscribed consumers. For instance, if a service channel has 3 subscribed consumers and a message from a service producer is only received by two of these consumers, availabilityPerEvent would be 0.66;
- individualLatency is a metric of type latency and measures the latency for each message received by each client in milliseconds.
- medianLatency is a derived metric of type latency and computes the median latency of the messages received within one hour per each client.

For simplicity, the provided SLA defines just one guaranteeTerm with one scope (i.e. the part of the system to which the SLOs apply). In this guaranteeTerm, 3 SLOs comprising the previously defined metrics are defined using simple boolean expressions.

This concrete syntax has been implemented using the Xtext Framework. One of the key advantages of using Xtext is that it is capable of automatically providing all the necessary tooling to support the development of documents following the defined syntax.

As a result, the extended AsyncAPI Toolkit provides a content-assisted editor and parser to define SLAs along with the AsyncAPI specification. Figure 3 provides a screenshot of the tool. The AsyncAPI Toolkit with SLA support is now officially available on GitHub (see footnote 1).

6 CONCLUSIONS

In this work we have presented a framework to specify service-level agreement documents for asynchronous services. Among its different assets, we have proposed (i) a quality model aligned with the ISO/IEC 25010 standard that defines several quality metrics for asynchronous services; (ii) a metamodel and concrete syntax to define SLAs for such services, named AsyncSLA, and aligned with the WS-Agreement and AsyncAPI specifications, that enables the
The research described in this paper has been partially supported by the automation of SLA related activities, such as the automatic generation of monitors from the SLA. We also plan to analyze how to better integrate our proposed DSL to existing WoT standards and to implement additional features in the AsyncAPI Toolkit to support this new DSL.

As future work, we plan to extend and validate our quality model and to implement additional features in the AsyncAPI Toolkit for the automation of SLA related activities, such as the automatic generation of monitors from the SLA. We also plan to analyze how to better integrate our proposed DSL to existing WoT standards and validate it through its deployment in a real-world company, serving as a comprehensive case study.

ACKNOWLEDGEMENTS

The research described in this paper has been partially supported by the AIDOaRT and TRANSACT projects, which have received funding from the ECSEL Joint Undertaking (JU) under grant agreement No. 101007350 and 101007260 respectively. The JU receives support from the European Union’s Horizon 2020 research and innovation programme and Sweden, Austria, Czech Republic, Finland, France, Italy, Spain, Netherlands, Germany, Poland, Belgium, Denmark, and Norway.

REFERENCES