Fuzzy Logic As-a-Service for Ambient Intelligence Environments

Amir Pourabdollah, *Member, IEEE,* Christian Wagner, *Senior Member, IEEE,* Giovanni Acampora, *Senior Member, IEEE* and Ahmad Lotfi, *Senior Member, IEEE*

***Abstract*—Fuzzy Logic Systems (FLSs) are normally associated with dedicated hardware/software systems. However, the distributed and pervasive architecture of many modern hardware/software systems is driving increasing interest in pervasive, distributed FLSs. Achieving this vision will require the design of FLS implementations which support client-server models and more specifically, cloud-computing and service-oriented solutions. Here, FLSs become a globally accessible service that enables openness, device independence, load balancing, resource sharing and ultimately cost effectiveness. In this paper, the recently standardised fuzzy mark-up language (IEEE-1855) and proposed extensions are used for designing Web Services for FLS computations. The novelty of this approach is in integrating different FLS components (input collection, processing and output) into a single web service platform which uses a well specified language for communication over the Web via HTTP request/responses. The utility of this approach is shown in the context of implementing FLSs in Ambient Intelligent Environments.**

I. Introduction

Advances in Ambient Intelligence (AmI) and intelligent environments require artificial intelligence tools and techniques to be accessible by a large number of networked devices, ranging from small and low-power sensors to embedded devices to PCs and large servers. The distributed architecture that is preferred for the ambient computing systems implies that the sensors, processes and output devices/actuators can be physically distributed in the environment. Fuzzy Logic Systems (FLSs), are potentially considered to be computationally intensive, particularly when it comes to multi-input, multi-output and multi-rule FLSs [1]. Moreover, recent advances in Type-2 and non-singleton fuzzy logic systems requires even more computational power than standard type-1 systems [2]–[6]. Many designers avoid designing advanced forms of FLSs because of the increased complexity. While small or embedded devices, such as wearable or pervasive devices, are specially designed for networking and data communication, they are not capable of undertaking intense computation.

A common solution for the case of non-mobile computer systems is to offload the complex logic from the clients to specialised servers in a client-server model. However, recent advances in Cloud Computing, the Internet of Things (IoT) and Service-Oriented Architectures (SOAs) have led to a move from the classical client-server model to a service-oriented model for ambient intelligence [7], [8]. By applying this approach for FLSs, fuzzy logic can be viewed as-a-service, which, if held in a cloud-based architecture, enables for computation to be offloaded and hidden from the devices in an ambient intelligent environment.

Amir Pourabdollah and Ahmad Lotfi are with the School of Science and Technology, Nottingham Trent University, Nottingham, UK. Christian Wagner is with Lab for Uncertainty in Data and Decision Making (LUCID), at the School of Computer Science, University of Nottingham, UK. Giovanni Acampora is with University of Naples Federico II, Napoli, Italy. (email: amir.pourabdollah@ntu.ac.uk, christian.wagner@nottingham.ac.uk, giovanni.acampora@unina.it and ahmad.lotfi@ntu.ac.uk)

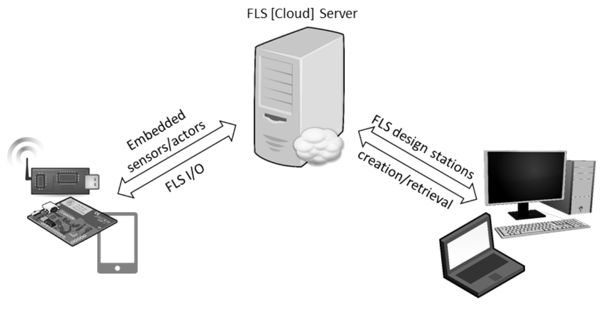


Fig. 1. The outline of a cloud-based model for distributed FLS computations.

The aim of this paper is to introduce a service-oriented model for FLS computations so that design and implementation of complex FLSs become more practical and cost effective. As will be shown, designing distributed architectures or client-server models for FLSs is a fairly new area with only very limited advances. The novelty of this paper is to move towards an open service-oriented design. This openness can be achieved by designing the system to be totally web-based and device-independent, particularly establishing data exchange formats which are both standardised and as human-readable as possible.

The application of this approach is not only limited to embedded systems and intelligent environments, but also to any situation where the processing logic of a FLSs needs to be abstracted from its data and presentation logics. For example, an individual client computer can define its required FLS (fuzzy sets, rule-base, etc.) on a server. Them, the same or even another client computer(s) can provide input data for the defined system, and finally the same or other client(s) can retrieve the computed output as illustrated in Fig. 1.

The main standard for implementing such an architecture is a web-based data language for FLS definitions [9]. The IEEE Standard 1855 (2016) [10] for Fuzzy Markup Language (also known as FML) is the current standard for such a purpose. IEEE-1855 is an XML-based data language that allows modelling a FLS in a human-readable and hardware independent way. In this paper IEEE-1855 will be used as the basic design standard and the extensibility of this standard is a key in developing the architecture.

Following this introduction, related work will be reviewed in Section II in order to clarify the position of this paper. Then the requirements for a service-oriented architecture of FLSs, the proposed model and the details of a sample implementation will be explained in Section III. Finally, the paper is concluded and future research directions are outlined in Section IV.

II. Background

A. Motivation

The motivation behind a service-oriented solution for FLSs is based on the potential and increasing interest in employing FLSs in an AmI and distributed context. Several specific aspects are reviewed below.

1) Distributed Architecture of Ambient Intelligence: In the ambient intelligence paradigm, the components of a FLS are physically distributed in the environment designed for ambient intelligence. In such an environment, sensors which collect the input data for the FLS, processors which execute the FLS and output devices (such as actuators) are distributed within and beyond the environment. From an efficiency, flexibility and redundancy point of view, it is desirable for not only input/output devices to be distributed, but that also the required processing power be distributed among a number of servers. This paradigm requires a dynamic balance of tasks between the three components which is not possible with fixed hardware/software designs.

2) High-Power Computation Requirement: The computational power is a known bottleneck for FLS implementations (e.g. the complexity of type-2 FLS defuzzification [2]). The complexity of FLSs can increase drastically when the number of inputs, outputs and rules are increased. Many advanced FLS architectures, such as non-singleton or type-2 FLSs are not used because of their high computational complexity. A cloud-based solution that can dynamically allocate memory and processors to a particular FLS application is preferred over fixed hardware solutions.

3) Reuse of Computation Results: In order to avoid repeated computations, a server can keep track of the inputs coming from, and the output sent to, different input/output devices for a single FLS. For example, if the inputs from a device A have been already processed in the (recent) past, the saved outputs can be used as a response (i.e. using a lookup table). Moreover, the definition of a single FLS for a particular application may be shared and reused by different other applications in some other environments. This however is only be possible if a FLS servers can be queried for FLSs' definitions and their input/output history.

4) Openness and Accessibility: This is a general motivation for any type of FLS application, not limited to ambient intelligence. Currently, the available tools for FLS computations are mostly fitted to single-station purposes. For example, special tools and libraries are developed for fuzzy logic computation in MATLAB and R (e.g. [11], [12]). A web-based solution where the FLS server is available on a web-server can help users through platform-independence and avoiding the need to use/install certain tools or programming languages for FLS computations. This is particularly relevant for research or educational purposes where access to and sharing of both resources and results are important. As an example, JuzzyOnline [13] provides a browser-based GUI for FLS design and computation, avoiding the need for special tool availability on the client-side. For its server-side programming, it uses Juzzy [14] which is an open-source Java library for FLS computation, making it particularly suitable for the Web programming purposes.

5) Advances in Cloud Computing and the Internet of Things (IoT): The cloud Computing can be considered as a useful extension of the fixed-hardware client-server architecture, in order to make the system resources more flexible to be allocated, depending on the needed computational power for a particular FLS. This will make the FLS computation as-a-service that can be accessed from any devices. Having this accessibility range, an FLS server can also be used in the context of the IOT, by any IOT device located at any geographical location.

B. IEEE-1855 (Fuzzy Mark-up Language)

Introduced in [15] and standardised in [10], IEEE-1855 is an XML-based language that allows modelling a FLSs in a human-readable and hardware independent way. We do not review its features and the range of applications here (a complete coverage can be found in [16]). Prior to this, a part of IEC 61131 [17] which was dedicated to fuzzy controllers could be used for defining a limited range of fuzzy logic systems (also called Fuzzy Control Language - FCL).

As far as server-side programming is concerned, IEEE-1855 has the great advantage of being directly convertible to programming logic. Using an Extensible Stylesheet Language Translator (XSLT), a FLS described in IEEE-1855 can be instantly transformed to a number of different programming language codes (e.g. Java) [18] so minimal server-side effort is required for encoding a FLS definition in local program logic. IEEE-1855 even allows an FLS definition to be controlled from different locations by different agents, interacting with the environment, as shown in [19]. As we will see in this paper, although IEEE-1855 can be used for defining a fuzzy logic system, it does not fully cover all of the proposed model's requirements. Some additional data elements need to be developed on top of the standardised language for purposes such as exchanging input/output data to or from the defined FLS. The extensibility of this language conveniently provides the possibility to add the additional required data elements.

C. Related Works on Distributed FLS Architectures

A number of research papers have focussed on Service-Oriented Architectures (SOAs) for FLSs. A group of software architectures under the name of Collaborative Architecture for Fuzzy Modelling are reviewed in [20]. In another SOA approach to fuzzy computing, a similar solution is proposed for Distributed Fuzzy Decision Making System (D-FDMS) purposes [21]. Also in [22] a multi-agent method is developed for neuro-fuzzy computations based on the IEEE-1855. In the mentioned works, the proposed architectures are either not based on a totally service-oriented paradigm or not on a standardised data exchange format.

This paper in part builds on the proposal of ubiquitous fuzzy computing for AmI presented in [23]. Here, a distributed service-oriented architecture is proposed having four subsystems for design, run-time, service retrieval and fuzzy-controlled AmI environments. IEEE-1855 is used as a data exchange language for describing a FLS in design environment, but the communication format between the other subsystems, particularly between the fuzzy-controlled AmI and run-time environments is based on direct TCP/IP socket communications in a user-defined data exchange format.

In order to make the architecture components less coupled and the data language more device-independent, open and human-readable, this paper proposes a novel approach where all data communications within the service-oriented architecture are web-based and exchanged via HTTP in XML format.

III. A New Service-Oriented Architecture

A service-oriented architecture (SOA) is defined as a collection of self-contained, networked, loosely-coupled and reusable software components, that are commonly used by clients with no or minimum dependency in respect to their hardware or software platforms [24], [25]. In a SOA for FLSs, the main services are distributed among one or more servers being contacted by a number of clients. In the following subsections, firstly the main required functionalities for such an architecture are discussed together with the proposed solutions for each one, then the extended IEEE-1855 schema and a sample implementation of the solution are explained.

A. Required Functionality

A number of tasks are considered as the main functional requirements for developing the solution. For each function, a Web service invocation through API is designed which includes a request and a response. A summary of these invocations is shown in Fig. 2 as a sequence of requests/responses between the client and the server sides. Since the FLS design parameters, inputs and outputs are being reused in the system, it is necessary to store them in a database. It is also noticeable that the solution is currently limited to type-1 singleton FLSs. The other FLS types will be considered as part of future work.

1) Creation of a Fuzzy Logic System: This function – *CreateFLS* – is required when a FLS is designed for the first time or edited by the client(s) which are in charge of the FLS design. The pieces of information that are required to be stored for a particular FLS include the variables for input(s) and output(s), the associated fuzzy terms including fuzzy set parameters for each term, the fuzzy rule-base including rules' antecedent(s) and consequent(s), and methods of inference and defuzzification for each output.

IEEE-1855 can be used for this purpose since its standard XML schema (listed in [10]) allows the above pieces of information to be exchanged. A sample FLS definition is listed below for the known example of a restaurant tipping problem. Note that namespaces in XML tags are not used in the following listings for the sake of simplicity.

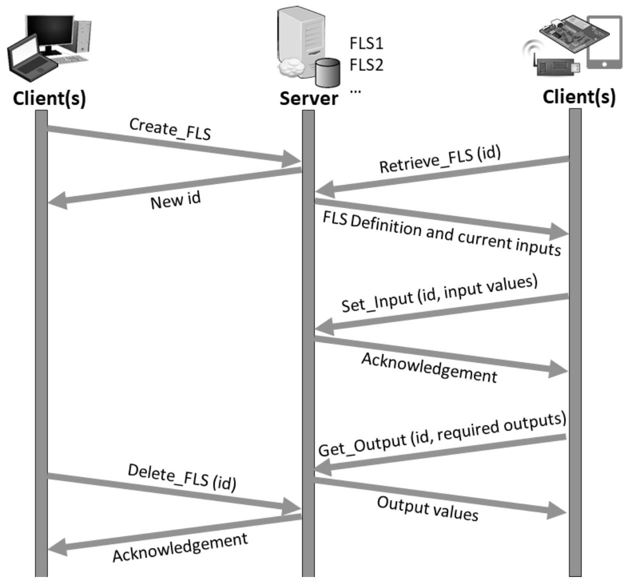


Fig. 2. The sequence diagram of the five types of client-server communications through the API. The clients shown on the left and the right to illustrate their logical roles, however they can be the same or different client in practice. The left-side clients are those used by FLS designers (e.g. human-operated PCs) and the right-side ones usually have an I/O role (e.g. embedded systems).

```
<FuzzyController name="newSystem" ip="localhost">
 <KnowledgeBase>
  <FuzzyVariable name="food" domainleft="0.0"
  domainright="10.0" scale="" type="input">
   <FuzzyTerm name="badFoodMF" complement="false">
    <TriangularShape Param1="0.0" Param2="0.0"
    Param3="10.0" />
   </FuzzyTerm>
   ... [other fuzzy variables/terms for input here]
  </FuzzyVariable>
  <FuzzyVariable name="tip" type="output"
  scale="Euro" domainright="30.0" domainleft="0.0"
  accumulation="MAX" defuzzifier="COG"
  defaultValue="0.0">
   <FuzzyTerm name="lowTipMF" complement="false">
    <GaussianShape Param2="6.0" Param1="0.0"/>
   </FuzzyTerm>
   ... [other fuzzy variables/terms for output here]
  </FuzzyVariable>
 </KnowledgeBase>
 <RuleBase name="Rulebase1" activationMethod="MIN"
 andMethod="MIN" orMethod="MAX" type="mamdani">
  <Rule name="reg1" connector="and" operator="MAX"
  weight="1.0">
   <Antecedent>
    <Clause>
     <Variable>food</Variable>
     <Term>badFoodMF</Term>
    </Clause>
    ... [other rule’s clauses here]
   </Antecedent>
   ... [other antecedents here]
   <Consequent>
    <Clause>
     <Variable>tip</Variable>
     <Term>lowTipMF</Term>
    </Clause>
    ... [other rule’s clauses here]
   </Consequent>
   ... [other consequents here]
  </Rule>
  ... [other rules here]
 </RuleBase>
</FuzzyController>
```

In response, the server creates/recreates a FLS (i.e. makes/edits a record of the FLS in the database) and assigns a unique id to it, then acknowledges the FLS creation by sending back the id to the client together with a confirmation message. A sample response is shown below.

```
<FuzzyController>
 <FLSResponse id="55" type="CreateFLS">
  <Message>FLS is created successfully.</Message>
 </FLSResponse>
</FuzzyController>
```

The XML schema of the above response (and the responses shown in the next functions) is different from IEEE-1855 schema. The schema extension discussed in the next subsection will show how the newly required elements are dealt with.

2) Query Fuzzy Logic System: Different types of clients may need to retrieve the information stored on the server about the status of a FLS. The FLS's status includes its existence and the parameters or the current input values stored for it. This functionality is implemented as a multi-purpose invocation named as *QueryFLS*. Having a particular FLS id, The client may request for the FLS design and/or for the last collected input(s) to the system to be retrieved (also see the SetInput description below). A sample QueryFLS request is listed here:

```
<FuzzyController>
 <FLSRequest id="55" type="QueryFLS"/>
</FuzzyController>
```

If the requested FLS exists within the server's database, the server response will includes the FLS definitions (as already defined in IEEE-1855 by *CreateFLS*) in addition to the last stored inputs (if already defined by *SetInput*). It is valid to query a FLS which is already defined but where one or more inputs are not set yet, in which case no input will be returned for the unknown inputs. A sample response is:

```
<FuzzyController>
 ... [IEEE-1855 elements for FLS description]
 <FLSResponse id="55" type="QueryFLS">
  <Input name="food">8</Input>
  <Input name="service">9</Input>
  ... [same for all the known inputs]
 </FLSResponse>
</FuzzyController>
```

3) Set Input: The clients must be able to set individual values for the FLS inputs. It is also expected that a single invocation by a particular client device may include any number of inputs. This will balance the data collection load by enabling the system to collect the individual or grouped inputs from different devices. The clients invoke *SetInput* to send one or more input values to the server. The design ultimately enables for inputs to be set either as a fuzzy set (i.e. a non-singleton input) or a crisp value, however for simplicity in this paper, we consider the inputs to be crisp values. A sample listing is provided here:

```
<FuzzyController>
 <FLSRequest id="55" type="SetInput">
  <Input name="food">8</Input>
  ... [same for all the known inputs to the device]
 </FLSRequest>
</FuzzyController>
```

If the inputs are valid, the server will store the input values in the database, and will send an acknowledgement message to the client, similar to the *Output* element introduced by *CreateFLS*.

```
<FuzzyController>
 <FLSResponse id="55" type="SetInput">
  <Message>Input(s) are set successfully.</Message>
 </FLSResponse>
</FuzzyController>
```

Any collected input can be stored in the database but only the last set will be used during FLS execution. In future, additional functions will enable the deletion of unwanted historical input/output data based on user request or a timer, without affecting FLS definitions.

4) Get Output: The client devices must also be able to receive the output value(s) of the FLS by request. It is preferred that the server(s) can send back the requested output either as a fuzzy sets or as a defuzzified value, however for simplicity, in this paper we limit the requirement to defuzzified values. A sample request is as follows:

```
<FuzzyController>
 <FLSRequest id="55" type="GetOutput">
  <Output name="tip" />
  ... [same for all the required outputs]
 </FLSRequest>
</FuzzyController>
```

Being able to request any number of output values practically means that the system can avoid unnecessary computation for unwanted outputs. In a complex FLS computation scenario, the same FLS can be defined on a number of servers, but each server may provide a part of the output.

In this scenario, the sensors may send their data to multiple servers but an output device may selectively request its required data from a single server - while the other servers are busy providing data for other output devices. This scenario has been illustrated in Figure 3.

Also as mentioned before, the system has access to the previous FLS runs, so that it can avoid repeating the computation if the same FLS has already been run with the same inputs in the past. If the requested output can be either calculated or retrieved, the system will respond to the client, similar to the following listing:

```
<FuzzyController>
 <FLSResponse id="55" type="GetOutput">
  <Output name="tip">8.4555</Output>
   ... [same for all the required outputs]
  </FLSResponse>
</FuzzyController>
```

5) Deletion of Fuzzy Logic System: Finally, the clients must be able to request for removing a FLS from the list of the defined FLSs in the database, if neither the FLS definition nor its past input/output history are needed anymore. The system may optionally be designed to remove the FLSs after a certain inactivity lifetime.

A sample request for deleting a particular FLS indicated by its id is as follows:

```
<FuzzyController>
 <FLSRequest id="55" type="DeleteFLS" />
</FuzzyController>
```

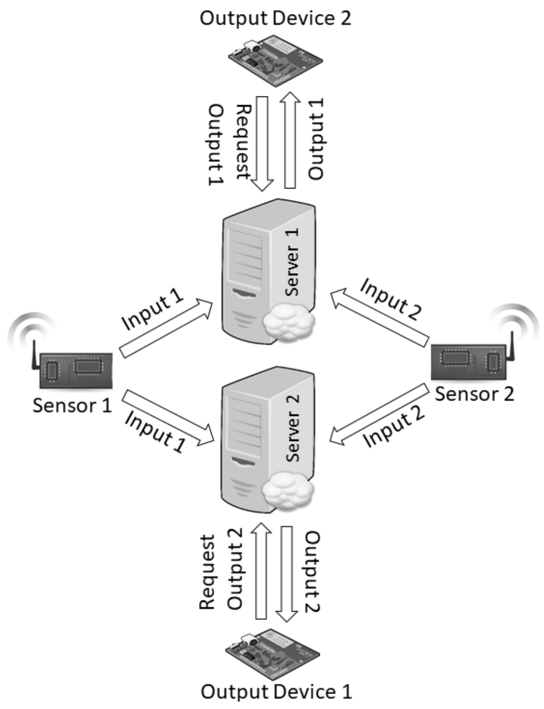


Fig. 3. In a complex computation scenario, output devices may request data from different servers which has the same FLS definition stored. The servers receive input data from all the sensors but process them according to the requested output. This will balance the processing load by specialising the output generation.

Additionally, this is an example of the system response, if the deletion is successful:

```
<FuzzyController>
  <FLSResponse id="55" type="DeleteFLS">
    <Message>The FLS is deleted successfully</Message>
  </FLSRequest>
</FuzzyController>
```

B. Extending IEEE-1855 Schema

The XML schema of the requests and response described previously is different from IEEE-1855 schema. It is noticeable that the newly added XML elements are inside the two main elements *FLSRequest*, *FLSResponse* which are deliberately put inside the IEEE-1855's root element (*FuzzyController*). This means that *FLSRequest*, *FLSResponse* and their sub-elements are candidate extensions to the original IEEE-1855 schema. It also means that all the requests and responses to/from the server can be validated using a single schema.

By merging the schema for the new required elements and the standard IEEE-1855 schema, an extended schema is produced. This schema has the same root element from the original schema (*FuzzyController*). In addition to the original sub-elements (such as *KnowledgeBase*, *RuleBase* etc. as defined in [10]), two additional complex-type elements for handling FLS requests and responses are added (*FLSRequest*, *FLSResponse*) under the root element. According to the requirements discussed in this section, the extended IEEE-1855 schema is designed as listed in Figure 4.

C. A Sample Implementation

Following the above design, a cloud-based server has been developed that can send and receive the required HTTP messages to/from clients. The cloud server is provided by Microsoft Azure so that the computational resources, including processors, hard disk and memory can be dynamically allocated or released based on the requirements and FLSs' complexities. The server uses Apache Tomcat as a standard implementation of Java Servlet. The server is programmed in Java, and for fuzzy logic computation, the open-source Java library (Juzzy [14]) is used. This server-side setting can fit to a wide range of applications, particularly where the client-side agents are sensors and/or actuators in ambient intelligent environments.

For testing, a web-based application is developed on the same server that handles the server input/output tasks in a GUI. Finally, for storing the FLS definitions, inputs and outputs, PostGreSQL is used as a general-purpose relational database. During the test, the server was able to serve the HTTP requests coming from different clients. The test was done using PCs and smart phones as clients and based on a simple FLS application.

The test and its results are limited to a specific exemplar scenario (the same example that is used in the provided listings through this paper). It has been used as a proof-of-concept in this paper, so that in future works the proposed method will be applied in a practical problem, in which more comprehensive testing will be done for real-world scenarios where real sensors/output devices are being set up.

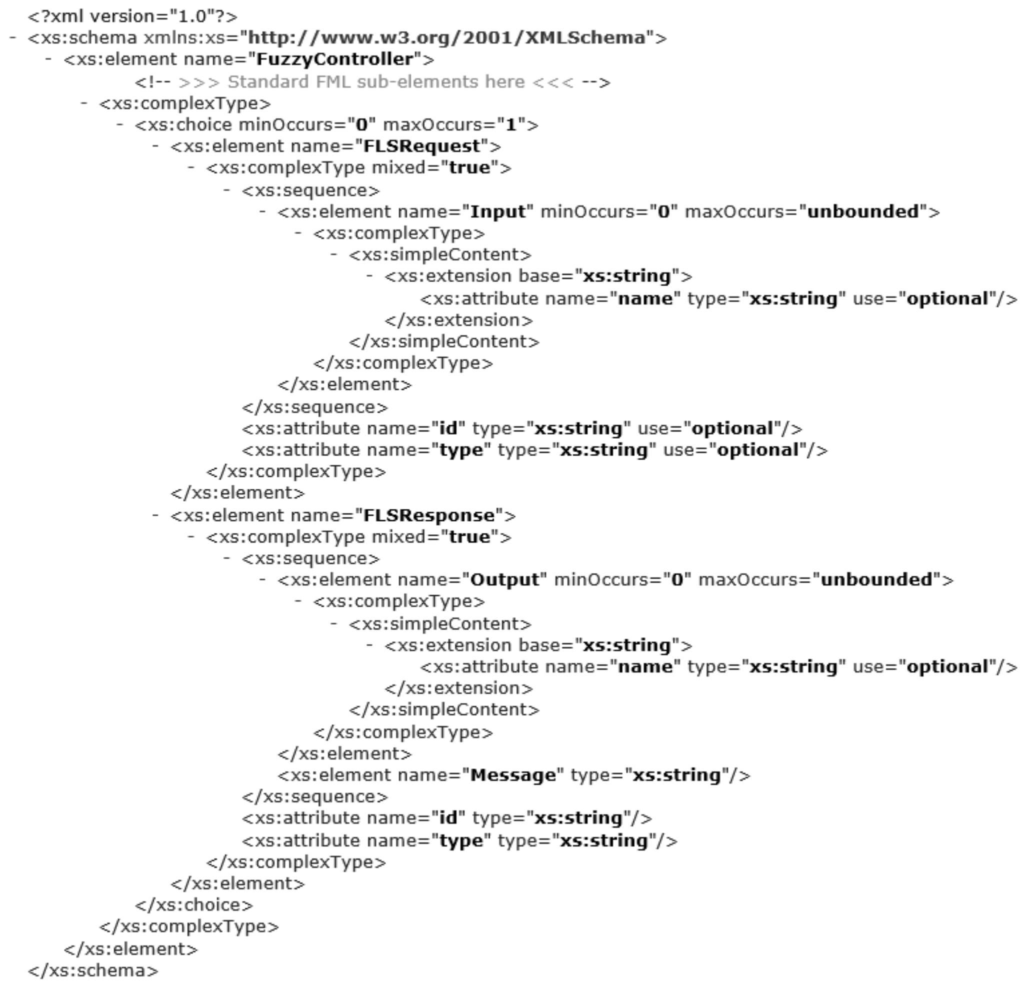
IV. Conclusions and Future Works

This paper provides a development road map for a web-based service-oriented FLS architecture, as an example of adopting the newly approved IEEE-1855 Standard. Although the architecture is showcased in the context of AmI environments, it can be applied to a much wider area of applications, i.e., anywhere the processing logic of a FLSs needs to be abstracted from its data and presentation logics.

The main motivation is enabling the flexible distribution of potentially complex computation required for FLSs from the clients (sensors, actuators, embedded systems, etc.) to dedicated servers. In particular, using virtualised cloud services provides elasticity to the solution, effectively enabling fuzzy logic as-a-service that is accessed universally.

Sharing resources, hardware/software independence, reuse, load balancing among FLS devices and ultimately cost effectiveness are the other advantages of such an architecture. IEEE standard 1855-2016 [10] as the standardised data exchange format for FLSs, has been used and extended in this proposal, enabling not only data exchange related to FLS definitions, but to also handle all requests and responses exchanged between clients and servers, e.g. collecting inputs and distributing outputs.

There are several aspects of extension for the proposed system from a technical point of view and we encourage the FLS community to engage and provide feedback with the development, both in terms of prioritisation of features, but also in supporting the collaborative development effort.



```
  <?xml version="1.0"?>
- <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  - <xs:element name="FuzzyController">
          <!-- >>> Standard FML sub-elements here <<< -->
    - <xs:complexType>
      - <xs:choice minOccurs="0" maxOccurs="1">
        - <xs:element name="FLSRequest">
          - <xs:complexType mixed="true">
            - <xs:sequence>
              - <xs:element name="Input" minOccurs="0" maxOccurs="unbounded">
                - <xs:complexType>
                  - <xs:simpleContent>
                    - <xs:extension base="xs:string">
                        <xs:attribute name="name" type="xs:string" use="optional"/>
                      </xs:extension>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
              </xs:sequence>
              <xs:attribute name="id" type="xs:string" use="optional"/>
              <xs:attribute name="type" type="xs:string" use="optional"/>
            </xs:complexType>
          </xs:element>
        - <xs:element name="FLSResponse">
          - <xs:complexType mixed="true">
            - <xs:sequence>
              - <xs:element name="Output" minOccurs="0" maxOccurs="unbounded">
                - <xs:complexType>
                  - <xs:simpleContent>
                    - <xs:extension base="xs:string">
                        <xs:attribute name="name" type="xs:string" use="optional"/>
                      </xs:extension>
                    </xs:simpleContent>
                  </xs:complexType>
                </xs:element>
                <xs:element name="Message" type="xs:string"/>
              </xs:sequence>
              <xs:attribute name="id" type="xs:string"/>
              <xs:attribute name="type" type="xs:string"/>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
```

Fig. 4. The extended IEEE-1855 schema (in XSD) that can be used to validate all of the web-based input/output data exchanges in the proposed architecture.

Practically speaking, the system will be evaluated for real-world FLS applications. This will give us the opportunity to refine the proposed schema and/or the API invocation formats, and to provide more in-depth technical recommendations for implementing the architecture elsewhere.

It is noticeable that only a particular case of fuzzy logic systems (namely, the rule-based systems) are addressed in thie paper. In the future, other fuzzy services, such as fuzzy querying over fuzzy databases or fuzzy ontologies may be considered. Additionally, because of the close relation between fuzzy mark-up language and fuzzy ontologies, extending the web services to the Semantic Web will be an interesting option.

Finally, the work will lead to contributing to the development of the next IEEE-1855 edition, not only by incorporating the proposed extended schema, but also by including more advanced FLS variations in the next schema release: non-singleton and type-2 FLSs.

V. Acknowledgment

The authors would like to thank Mrs. Divya Rao for her contribution to the system development discussed in this paper. Also, the valuable comments from anonymous reviewers are highly appreciated.

References

[1] Y. H. Kim, S. C. Ahn, and W. H. Kwon, "Computational complexity of general fuzzy logic control and its simplification for a loop controller," *Fuzzy Sets and Systems*, vol. 111, no. 2, pp. 215–224, 2000.

[2] R. John and S. Coupland, "Type-2 fuzzy logic: A historical view," *IEEE computational intelligence magazine*, vol. 2, no. 1, pp. 57–62, 2007.

[3] G. C. Mouzouris and J. M. Mendel, "Nonsingleton fuzzy logic systems: theory and application," *IEEE Transactions on Fuzzy Systems*, vol. 5, no. 1, pp. 56–71, 1997.

[4] C. Wagner, A. Pourabdollah, J. McCulloch, R. John, and J. M. Garibaldi, "A similarity-based inference engine for non-singleton fuzzy logic systems," in *2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, July 2016, pp. 316–323.

[5] A. Pourabdollah, C. Wagner, J. H. Aladi, and J. M. Garibaldi, "Improved uncertainty capture for nonsingleton fuzzy systems," *IEEE Transactions on Fuzzy Systems*, vol. 24, no. 6, pp. 1513–1524, Dec 2016.

[6] A. B. Cara, C. Wagner, H. Hagras, H. Pomares, and I. Rojas, "Multiobjective optimization and comparison of nonsingleton type-1 and singleton interval type-2 fuzzy logic systems," *IEEE Transactions on Fuzzy Systems*, vol. 21, no. 3, pp. 459–476, June 2013.

[7] J. Cubo, A. Nieto, and E. Pimentel, "A cloud-based internet of things platform for ambient assisted living," *Sensors*, vol. 14, no. 8, pp. 14 070–14 105, 2014.

[8] V. Issarny, N. Georgantas, S. Hachem, A. Zarras, P. Vassiliadist, M. Autili, M. A. Gerosa, and A. B. Hamida, "Service-oriented middleware for the future internet: state of the art and research directions," *Journal of Internet Services and Applications*, vol. 2, no. 1, pp. 23–45, 2011.

[9] B. N. Di Stefano, "On the need of a standard language for designing fuzzy systems," in *On the Power of Fuzzy Markup Language*. Springer, 2013, pp. 3–15.

[10] "IEEE standard for fuzzy markup language," IEEE Std 1855-2016, pp. 1-89, May 2016.

[11] J. R. Jang, *MATLAB: Fuzzy logic toolbox user's guide: Version 1*. Math Works, 1997.

[12] C. Wagner, S. Miller, and J. M. Garibaldi, "A fuzzy toolbox for the R programming language," in *Fuzzy Systems (FUZZ), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1185–1192.

[13] C. Wagner, M. Pierfitt, and J. McCulloch, "Juzzy online: An online toolkit for the design, implementation, execution and sharing of type-1 and type-2 fuzzy logic systems," in *Fuzzy Systems (FUZZ-IEEE), 2014 IEEE International Conference on*. IEEE, 2014, pp. 2321–2328.

[14] C. Wagner, "Juzzy - a java based toolkit for type-2 fuzzy logic," in *Advances in Type-2 Fuzzy Logic Systems (T2FUZZ), 2013 IEEE Symposium on*. IEEE, 2013, pp. 45–52.

[15] G. Acampora and V. Loia, "Fuzzy markup language: A new solution for transparent intelligent agents," in *Intelligent Agent (IA), 2011 IEEE Symposium on*. IEEE, 2011, pp. 1–6.

[16] G. Acampora, V. Loia, C.-S. Lee, and M.-H. Wang, *On the power of fuzzy markup language*. Springer, 2013.

[17] "IEC Standard for Programmable controllers," IEC 61131-7 Part 7: Fuzzy control programming, 2000.

[18] G. Acampora, "Fuzzy Markup Language: A XML Based Language for Enabling Full Interoperability in Fuzzy Systems Design," *On the Power of Fuzzy Markup Language*, vol. 296, pp. 17–31, 2013.

[19] G. Acampora, V. Loia, and A. Vitiello, "Distributing fuzzy reasoning through fuzzy markup language: An application to ambient intelligence." 2013.

[20] W. Pedrycz, "Collaborative architectures of fuzzy modeling," *Computational Intelligence: Research Frontiers*, pp. 117–139, 2008.

[21] K. K. Yuen and H. C. Lau, "Towards a distributed fuzzy decision making system," in *KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications*. Springer, 2008, pp. 103–112.

[22] G. Acampora, V. Loia, and A. Vitiello, "Using ANFIS and FML for deploying transparent services in smart environments," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*. IEEE, 2012, pp. 628–633.

[23] G. Acampora and V. Loia, "A proposal of ubiquitous fuzzy computing for ambient intelligence," *Information Sciences*, vol. 178, no. 3, pp. 631–646, 2008.

[24] L.-J. Zhang, J. Zhang, and H. Cai, "Service-oriented architecture," *Services Computing*, pp. 89–113, 2007.

[25] K. B. Laskey and K. Laskey, "Service oriented architecture," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 1, pp. 101–105, 2009.