Abstract: Fuzzy approaches offer a high potential to improve decision making within business applications. Nevertheless, they mainly exist as self-contained solutions that are not designed to collaborate with company-wide business application systems. Hence, they are often not brought into action as part of running business solutions. Web services are innovative distribution techniques, which are able to solve this problem. Based on an application-neutral representation of fuzzy data and rule-bases using the Extensible Markup Language (XML), we propose to make fuzzy control via Web services available. With this, once developed, fuzzy solutions can be offered to diverse companies, where they can be integrated into business application systems using standardized interfaces. Starting with a former fuzzy application, we show how to develop corresponding Web services and how to use them within a business application.

Key-Words: Fuzzy control; peer-to-peer architectures; XML; WSDL; UDDI; SOAP; credit scoring

1 Developing and Integrating Fuzzy Solutions

There are still obstacles that make it difficult to utilize fuzzy approaches for business applications as major problems may arise while integrating fuzzy applications into standard business application systems [9]. Integration problems concern combining fuzzy applications with each other, and integrating them into conventional business application systems, e.g. mismatched data types. As a matter of fact, fuzzy approaches mainly exist as self-contained solutions that are not designed to collaborate with company-wide business application systems, and thus, they are often not brought into action as part of running business solutions.

In [12] we proposed to use a common way to represent fuzzy information to soften the lack of integration between fuzzy applications and common business application systems as well as between fuzzy approaches that were developed on different development platforms. There, we introduced a formal syntax for important fuzzy data types. We made this syntax operable by defining appropriate Document Type Definitions (DTD), and we showed how fuzzy information, whose description is based on these DTDs, could be exchanged between application systems by means of the Extensible Markup Language (XML) [1].

On the other hand, we have already focused on techniques to develop fuzzy solution more efficiently [10]. There, we introduced a flexible Java-based fuzzy kernel, where business applications could be built on. Taking this approach as a basis, we enhanced the idea of the Java-based fuzzy kernel by subdividing the kernel into smaller software components [11]. By doing so, we focused on the problem of developing business applications using fuzzy techniques, as the use of software components simplifies the task of programming the application
itself, since, in most cases, no program has to be written at all.

The mentioned techniques work fine as far as fuzzy approaches, which need to be combined with each other or that need to be integrated into a company-wide business application system, are developed with fuzzy components. However, collaboration between fuzzy applications that were developed using different development environments may still lead to integration problems.

We propose to use Web services to diminish this kind of integration problems. Based on the above-mentioned application-neutral representation of fuzzy data and rule-bases, we make business logic within fuzzy components available as Web services. Once developed, fuzzy solutions can be offered to different companies, where they can be integrated into business application systems using standardized interfaces.

In the following, we explain services of software components offering fuzzy functionality as Web services. Starting with a fuzzy controller service, we show how to develop corresponding Web services and how to use them within a business application. The interaction between this kind of services bases on the recommendations given in [12], which we reformulated using XML Schema in order to improve validation.

2 Web Services

Using Web services means a change from traditional client-server to peer-to-peer architectures. Software components that semantically encapsulate discrete functionality are accessible over the Internet. The result is a network of distributed functions. Web services make complex business logic available worldwide. In order to integrate Web services into its own application architecture only the knowledge about the interface is required. For this reason, development and maintenance costs of building business applications are lasting reduced. The selection of a concrete service can ensue both at design time and runtime. Important criteria may be quality, costs, availability, performance, security, and reliability.

The Web service architecture describes three basic operations and organizational roles [6]. Fig. 1 illustrates the collaboration of these roles.

Beside the development of the software component the service provider uses the Web Service Description Language (WSDL) to specify deploying messages and data types of parameters and return values. WSDL is an XML Schema published by the World Wide Web Consortium (W3C). A WSDL document contains common information like service name and a human readable description of network services, generally. Besides, it specifies on the one hand the operations in an abstract way. On the other hand it binds these common messages and data types to protocols [2]. The strong separation between abstract definition and concrete network deployment makes possible that metadata are reused by different implementations.

![Fig. 1. Roles and operations in a Web service architecture](image)

The service provider uses the created WSDL document as basis for publishing the Web service in a Universal Description, Discovery and Integration (UDDI) business registry. UDDI comprises a protocol for describing the service providers themselves and their available Web services on the Internet. Leading software vendors provide appropriate repositories. They take the role of the service broker. The registries are connected to each other. Therefore, service providers have only to register at one business registry. Service requesters can query one of the available UDDI business registries to find a
Web service that fulfill their requirements. Browsing and editing are also possible.

After a service requester has discovered a suitable Web service, the last operation of the cycle is performed. The requester binds the located Web service to its supported communication mechanism for invoking.

From technical point of view the definition of a Web service consists of a set of standard protocols based on XML [5]. As well functional interfaces as data types are described in platform-neutral and programming language independent XML. Hence, offering services as Web services take the focus on protocols and not on implementations, because the Internet is composed of heterogeneous technologies.

The Web service Definition Language is used to describe the programmatic interface of Web services. Thus, WSDL is comparable with an Interface Definition Language (IDL) in other distributed environments. Generally, a service is a set of network endpoints that operate on messages. At first, operations and exchanging data are specified abstractly. Messages declare the exchanging data. So-called port types define a set of abstract operation definitions with input and output message elements. A binding XML element connects this metadata to a concrete network protocol and data format. The WSDL specification defines bindings to Simple Object Access Protocol (SOAP), Hypertext Transport Protocol (HTTP) and Multipurpose Internet Mail Extensions (MIME). For each definition a human readable documentation element can be inserted.

For publishing and finding Web services on the Internet a UDDI compliant business registry is used. The UDDI specification provides support for defining both: service and business information [4]. There are five basic data structures: businessEntity, businessService, publisherAssertion, bindingTemplate, and tModel. Fig. 2 shows the relationships between all of these data structures as class diagram modeled with the Unified Modeling Language (UML):

The Business entity represents the business that is offering the Web services belong to. The description consists of contact (e.g. phone, email, address) and categorization information. It is the top-level XML element for publishing information about a business and its Web service implementations.

![UDDI data model](image)

Business service and its binding templates specify the Web service. [3] gives detailed instructions for mapping from WSDL to UDDI. The business service entity contains the human-readable description. Nested binding templates structures specify how and where the Web service is accessed. Therefore, each binding template contains a reference to one or more instances of technology models (tModel), which define the technical interface of the service. Hence, the tModel references the messages and port types in form of a URL address to the corresponding WSDL interface document. Analogous to business entities category bags are used to classify technology models.

When publishing a Web service, the interface description must be available as a tModel before the service implementation is deployed as a business service. It is recommended to reuse an existing tModel in the registry. Each business service is assigned to a business entity.

Since UDDI Version 2.0 API specification so-called publisher assertions can be defined in order to model relationships between two business entities.
Both registration and lookup are done using SOAP. Appropriate SOAP messages are defined in an Application Program Interface (API) Specification [8]. Its unique key can identify as well business entities and services as binding templates and related technology models.

```xml
  <message name="PerformApproximateReasoningInput">
    <part name="ruleBase" element="types:RuleBase"/>
    <part name="factBase" element="types:FactBase"/>
  </message>
  <message name="PerformApproximateReasoningOutput">
    <part name="result" element="types:FactBase"/>
  </message>
  <portType name="FuzzyControllerServicePortType">
    <operation name="performApproximateReasoning">
      <input message="tns:PerformApproximateReasoningInput"/>
      <output message="tns:PerformApproximateReasoningOutput"/>
    </operation>
  </portType>
  <binding name="FuzzyControllerServiceSoapBinding" type="tns:FuzzyControllerServicePortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="performApproximateReasoning">
      <input soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/
      namespace="urn:fuzzy-controller-service" use="encoded"/>
      <output soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/
      namespace="urn:fuzzy-controller-service" use="encoded"/>
    </operation>
  </binding>
</definitions>
```

**Fig. 3. Interface description**

3 Fuzzy Controller as Web Service

For providing a fuzzy controller as Web service we have extended an implementation described in [11]. The functionality is accessible over a method `performApproximateReasoning` coded in a Java class `FuzzyController`. This method has got a fact- and a rule-base as input parameter. After the facts have been applied to the rules, the result returns as a fact-base.

As mentioned below, the service provider describes interface and implementation of a Web service in separated WSDL documents. Fig. 3 depicts the interface description.

After declaring several namespaces for different purposes the required fuzzy types `RuleBase` and `FactBase` are included by the import statement, because they are defined in another file that contains the appropriate XML Schema.

```xml
  <service name="FuzzyControllerService">
    <documentation>Fuzzy Controller Service using SOAP</documentation>
    <port name="FuzzyControllerServicePort" binding="fsi:FuzzyControllerServiceSoapBinding">
      <soap:address location="http://www.fuzzy-services.de:8080/soap/servlet/rpcrouter"/>
    </port>
  </service>
</definitions>
```

**Fig. 4. Implementation description**

For each direction for exchanging data a message XML element is defined. The first message specifies rule and fact base as parts of this message. The return value, a fact-base, is delivered by the second message. Within the port type XML element the operation `performApproximateReasoning` is composed with its input and output messages.

After the function is described abstractly, the first binding to concretes data exchange and communication protocols arises. Therefore, a binding XML element is defined, which references the specified port type. In this case, SOAP, in combination with HTTP, is used for communicating with the fuzzy controller.

The definition of the Web service ensues in a second WSDL document. Hence, it contains the concrete implementation of the specified inter-
Fig. 4 shows the content of this WSDL document.

After importing the interface document a Web service with the name FuzzyControllerService is declared. Besides the documentation, a port XML element connects this service to the SOAP binding. Finally, the location of the SOAP engine router must be specified as URL address.

The first implementation of this interface is based on the reference implementation of SOAP. It is the Tomcat SOAP engine from Apache [7]. For deploying SOAP based services a so-called deployment descriptor must be created (Fig. 5). It is a XML Schema that specifies identifier, implementing provider class, and type mappings.

```xml
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment" id="urn:fuzzy-controller-service" checkMustUnderstands="false">  
  <isd:provider type="java" scope="Request" methods="performApproximateReasoning">    
    <isd:java class="fuzzy.webservice.FuzzyController" static="false"/>
  </isd:provider>
  <isd:listener>org.apache.soap.server.DOMFaultListener</isd:listener>
  <isd:mappings>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:x="urn:fuzzy-controller-service" qname="x:FactBase" javaType="fuzzy.FactBase" java2XMLClassName="fuzzy.soap.apache.FactBaseSerializer"/>
    <isd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" xmlns:x="urn:fuzzy-controller-service" qname="x:RuleBase" javaType="fuzzy.RuleBase" java2XMLClassName="fuzzy.soap.apache.RuleBaseSerializer"/>
  </isd:mappings>
</isd:service>
```

Fig. 5. Deployment descriptor for Apache SOAP engine

Tomcat provides de-/serializer classes for simple data types and java beans. But the underlying implementation of the Fuzzy Controller does not allow the use of this serialization functions. For that reason, we developed special de-/serializers for common fuzzy types grouped in the package fuzzy.soap.apache.

4 Application Example

The Internet as a global and anonymous marketplace requires a reliable method for checking the creditworthiness of consumers. Therefore, creditors are often using a credit scoring model. Credit scoring is based on statistics. Information, such as bill-paying history, the number, type and age of accounts the consumer has, late payments and outstanding debt is the input for calculating a so-called credit score. Using a statistical program, creditors compare this information to the credit performance of consumers with similar profiles. The underlying model weights the factors in a different manner. Both criteria and their weights, and the exact computation procedures are closely guarded secrets of the companies that develop the scoring model. If the computed credit score lies within a particular range, the consumer is creditworthy.

Fig. 6. Fuzzy credit scoring model

Such credit scoring decision systems help companies minimizing their credit risks while maximizing profitability. But the credit score does not really measure how likely it is that the consumer will repay a loan and make payments according to the agreed-upon terms because both weights and ranges are crisp values. There is no difference between consumers within a range. Besides, scoring models disregard per-
sonal facts, which are often described linguistically (e.g. profession, occupational hazard, branch, education, hobbies and marital status), though companies collect them for a better customer relationship management. Assurance companies even consider such details for calculating insurance risk and premium. Why should creditors not use such information, too? The result would be a more detailed view on the living situation of the customer.

Therefore, we have developed an extended scoring model using fuzzy logic. Fig. 6 depicts selected linguistic variables and rules of our model.

To offer this kind of credit scoring we have developed a second Web service, which collaborates with the Fuzzy Controller Web service (fig. 7).

The service requester (e.g. owner of an internet shop) calls the credit scoring web service. The request contains names of linguistic terms as facts. It is aim of this Web service to put these values into a fact-base. Then, it forwards as well the fact-base as the rule-base to the fuzzy controller. The result returns as a fact-base. Finally, the credit score is defuzzified by the credit score Web service.

5 Conclusions

Collaboration between fuzzy applications that were developed using different development environments may still lead to integration problems. We propose to use Web services to diminish this kind of integration problems. With this, fuzzy solutions can be offered to different companies, where they can be integrated into business application systems using standardized interfaces.

Based on an application-neutral representation of fuzzy data and rule-bases, we showed how to make functionality of fuzzy components available as Web services. In addition, we gave a first application example, which use the developed fuzzy controller service. Providing a fuzzy calculator as Web service for mathematical operations on fuzzy numbers and intervals would be also possible.

Using Web services together with common Internet security techniques or payment scenarios (e.g. pay-per-use) are further topics being discussed in future.

References:

