A generic Web service is a set of operations accessible via specific end point. Each service operation is characterized with a signature. A signature is a collection of an operation’s name and two sets of types, instances of which are accepted and returned by the operation. The set of all signatures of a service’s operations is known as the interface to the service. A service’s interface defines the complete set of request messages that can be sent to the service.

The signature of an operation is not only a syntactic characteristic but also a semantic one and imposes restrictions upon the types and values of the operation’s parameters. This characteristic we will call the rigidity of an operation’s signature. Signature relaxation, in turn, is the loosening of this rigidity (i.e. the loosening of restrictions that the signature places on the types and values of parameters). Effectively, signature relaxation is the extension of parameters’ connotation and the set of their possible values.

Signature relaxation can be achieved by introducing a special sort of input parameters that will be used to define the meaning of other parameters. In this case the later parameters become controlled parameters and the former ones - controlling or identity parameters (they identify the semantics of other parameters).

We will call an operation with relaxed signature a generic operation. A generic interface (service), is an interface (service) that has at least one generic operation. A generic operation can be recognized via its identity parameters. If there is at least one parameter that defines or extends the semantics of another, then the operation is generic. If an operation does not have identity parameters it is conceived to be fine-grained.

B. Degree of Relaxation

Relaxed signatures are ambiguous because of identity parameters. Their semantics is unclear until exact values of identity parameters come at runtime. To measure this ambiguity we introduce the term degree of relaxation. Let an operation have one controlled parameter $p_1$ and two identity parameters $p_2$ and $p_3$. Assume that $p_2$ and $p_3$ have two possible values each$^1$. Then the same value of $p_1$ can be

---

$^1$The set of all possible values of a parameter is called the domain of the parameter.
connotated in four different ways. Thus, when a parameter \( p_j \) is controlled by \( l \) identity parameters each having the domain of size \( size_i, i = 1 \ldots l \), every value of \( p_j \) gets 
\[
 r(p_j) = \prod_{i=1}^{l} size_i \text{ different interpretations (one per each unique combination of identity parameters). We will call the value } r(p_j) \text{ the degree of relaxation of the parameter } p_j. \text{ The degree of an operation’s signature relaxation can be calculated as the sum of degrees of relaxation of each controlled parameter.} \]
\[
 r(o) = \sum_{j=1}^{m} r(p_j) = \sum_{j=1}^{m} \prod_{i=1}^{l} size_{ji}, \text{ where } m \text{ is the number of controlled parameters and } size_{ji} \text{ is the domain’s size of } i^{th} \text{ identity parameter. The higher the value } r(o), \text{ the more generic the operation is.} \]

C. The Value of Generic Web Services

When changing a Web service its provider has several options. (i) A new version of the service can be released in parallel to the old one. The provider will need to employ a versioning mechanism and maintain all versions of the service, which increases maintenance costs. (ii) The provider can change the service without maintaining backwards compatibility. Then clients will need to migrate their code, which creates additional costs for them. (iii) The provider may introduce an adapter or a converter that will compensate the difference between the versions of the service.

If a service has a generic interface adding new features does not require changing the interface\(^2\). Adding new features is supported by design via introducing new values of identity parameters. Adding these features does not require changing the signature of the service’s operations, because only the runtime aspects of the service is affected. This is exactly the value of generic interfaces. Service architects can greatly benefit from the fact that a generic operation can add functionality by simply extending the domain of its identity parameters. Thus, generic interfaces allow adding new functionality while preserving backwards compatibility.

The downside of generic Web services is the ambiguity of controlled parameters. The more values an identity parameter has, the greater the ambiguity of a parameter which it controls. This can result in less understandable service interface. Therefore, covering a too big piece of functionality with one operation is not recommended. In this case the ambiguity of an API will overwhelm the benefits of flexibility created by GWS.

D. Case Study

In this subsection we demonstrate the benefits of GWS by providing an example of how an interface could be relaxed and transformed to a generic one. This is illustrated on the following sales order related operations that were taken from Order-to-Cash SAP ES Workplace bundle:

1) Find SO Basic Data by Buyer & Delivery Blocked\(^3\)
2) Find SO Basic Data by Buyer & & Invoicing Blocked
3) Find SO Basic Data by Buyer & Incompleteness
4) Find SO Basic Data by Buyer & Basic Data
5) Find SO Basic Data by Buyer & Credit Block
6) Find SO Basic Data by Elements

The operations perform the same kind of functionality. They all search the backend system for sales orders. Each operation, however, uses different criteria. The operation 6 has the most advanced functionality. It supports the criteria of all other operations. The operation 4 supports the same set of criteria as the operation 6, but restricts the search to a single (specified) buyer party. The rest of the operations perform specific search based on two criteria that are reflected in the name of a corresponding operation.

One may conclude that the operations 6 and 4 are generic and the rest operations are fine-grained ones. This, however, is not true, because none of the operation has identity parameters. As can be seen all search criteria are hard coded as an operations\(^4\) input parameters.

Find SO Basic Data by Elements
\[
\text{string buyerNumber,} \\
\text{string material,} \\
\text{bool invoiceBlock,} \\
\text{bool creditBlock,} \\
\text{bool incompletion,} \\
\ldots \\
\]

Since no operation has identity parameters, the semantics of all parameters is defined at design time. Therefore, none of the operations has relaxed signature. All operations have difficulties with extending functionality. Basically, to support more search criteria SAP must either add new fine-grained search operation or add more parameters to the operation 6. In both cases changing the signature of the interface is required and backwards compatibility is not guaranteed.

To turn this interface into generic one we must relax the signature of at least one operation. Obviously, the best candidate for relaxation is Find SO Basic Data by Elements. The method can be relaxed in three ways.

1) Find SO Basic Data by Elements (  
\[
\text{string attribute,} \\
\text{string text} \\
\]

In this case the search can be performed only on a single criterion.

2) Find SO Basic Data by Elements (  
\[
\text{string[]} \text{ attribute,} \\
\text{string[]} \text{ text} \\
\]

This version of the operation searches on all attributes that a client specifies at runtime. All the criteria are used with a default predicate, for example and. A

\(^2\)At least that often as in case of fine-grained services.

\(^3\)SO stands for “Sales Order”.

\(^4\)Other operations follow the same principle, but have fewer parameters.
client application must guarantee that both input arrays are of equal size, otherwise a runtime error will be reported.

3) Find SO Basic Data by Elements {
   string[] attribute,
   string[] text,
   predicate[] predicate
}

This is the most relaxed version of the operation, because every text[i] parameter is controlled by two identity parameters attribute[i] and predicate[i]. As in the previous case a client application ensures the consistency of the input parameters.

The next question is which of the three versions to include to the interface. Obviously, all of them can be included. However, this is not necessary. We can include only the last two versions or even only the last one. The rest of the operations could be represented as overloads of the selected master methods.

There is also one interesting detail to be mentioned. The number of input parameters for Find SO Basic Data by Elements has been reduced. Now a client application only supplies the attributes on which search must be performed, where as in the non-relaxed version values of all parameters must be supplied no matter if they are used or not.

Another benefit of GWS that is seen from the example can be named as functionality aggregation. With a generic interface several fine-grained operations can be performed as a single whole. For instance, if we need to find all sales orders on three products, we need to execute three independent fine-grained searches on each of the products and then combine the results. With a generic sales order search this can be achieved with a single call like:

Find SO Basic Data by Elements {
   {"material","material","material"},
   {"iPod","iPhone","MacBook"}
}

Therefore, GWS can improve the performance by reducing the number of calls to a backend system.

III. RELATED WORK

Existing incompatibility resolution approaches for Web services are based on different kinds of mapping and message interception, which take place at different stages of message processing. The principle of communication mediator has been used in [1]. The paper suggested to pass incompatible Web service calls through a chain of adapters that compensates the difference in the interfaces of a client and a service. In [2] a similar technique is suggested. Incompatibility is reconciled inside a client-side proxy. In [3] incompatibility is resolved by applying xsl transformation to the raw xml content of messages.

Another breed of compatibility assurance approaches is based on semantic Web technologies [4], [5]. The main idea here is to enhance a Web service with a richer semantic description and introduce incompatibility resolution knowledge in terms of ontology axioms.

IV. CONCLUSION

The need for updating services creates a challenge to providers of maintaining compatibility between new and old service versions. This article contributes with the notion of generic Web service defined with the help of terms signature relaxation, identity parameter and degree of relaxation. Based on these notions an interface design technique was suggested. The technique helps service providers design Web services with more stable and extensible APIs.

The biggest value of generic Web services is that they allow for adding functionality by simply extending the domain of their identity parameters and without changing the signature of operations. The downside of generic Web services is the ambiguity of controlled parameters.

The future activities in the course of the research will concentrate on developing a method of finding appropriate degree of relaxation for an interface and verifying the theoretical results on a broader set of industrial Web services. In particular, a real-life example of inconsistent API evolution will be selected to see whether incompatibility could have been avoided if the interface had been created as a generic one.

REFERENCES


