February 15, 2008

RT0778
Security 8 pages

Research Report

Rule-based XML Mediation for Data Validation and Privacy Anonymization

Masayoshi Teraguchi, Issei Yoshida, Naohiko Uramoto

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Limited Distribution Notice
This report has been submitted for publication outside of IBM and will be probably copyrighted if accepted. It has been issued as a Research Report for early dissemination of its contents. In view of the expected transfer of copyright to an outside publisher, its distribution outside IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or copies of the article legally obtained (for example, by payment of royalities).

Rule-based XML Mediation for Data Validation and Privacy Anonymization

Masayoshi Teraguchi, Issei Yoshida, and Naohiko Uramoto
Tokyo Research Laboratory, IBM Research
1623-14, Shimotsuruma, Yamato-shi, Kanagawa-ken, 242-8502, Japan
{teraguti, issei, uramoto}@jp.ibm.com

Abstract

XML mediation for data validation and privacy anonymization of the very large complecated XML messages defined in some industry-specific specifications such as HL7 is becoming increasingly important in SOA because a lot of applications depend on various kinds of hard-coded data validation and privacy anonymization functions, which makes it difficult to keep consistency of the functions among the applications in SOA even when the schema of the XML messagses in the specifications is changed. This paper proposes a uniform rule-based approach to realize the functions as a XML mediation separated from the applications, which makes it easy to maintain consistency of the functions when they are updated in accord with the changes to the specifications. Therefore, application developers can readily utilize it with many applications in SOA. Our approach allows the developers to define a set of rules that consist of two components: constraint conditions in a conceptual data notation in the XML message and actions performed only when the conditions are satisfied. In order to make the rules independent from both the implementation-specific data notation and the industry-specific knowledge, we automatically transform the rules into the implementation-specific data representation using two more factors; one is the data mappings from the data notation in the rules to the concrete data representation in the implementation and the other is the implied data relationships hidden in the rules. It is very important to take into consideration a general way to import the implied knowledge because it often depends on the industry-specific data structure and it is usually given outside of the mediation system.

1. Introduction

Service-Oriented Architecture (SOA) is now emerging as the dominant integration framework in today's complex and heterogeneous enterprise computing environment. It promotes loose coupling of services so that Web services are becoming the most prevalent technology to implement SOA applications. Web services use a standard message protocol (SOAP [1]) to ensure widespread interoperability in the enterprise environment. The format of the application-specific data embedded into the SOAP body is typically simple and easy to understand. However there are some industry-specific specifications, such as HL7 [2] and XBRL [3], which define much more complicated data formats to satisfy requirements in those industries and which embed very large data objects directly into the SOAP body. Unfortunately since there is no good library currently available to check if the data in the SOAP body correctly confirms to the complex specifications or to check if the privacy-sensitive information in the data is sufficiently protected, application developers need to implement the functions of data validation and privacy anonymization by themselves in their each application deployed into SOA. But this would make it difficult to keep functional consistency among the applications when they need to update all functions separately implemented in the applications in responding to the changes to the specifications during the phase of application maintenance or upgrade.

In order to shift the workload away from the developers and to reduce the risk of the consistency problems, they can separate the functions of data validation and privacy anonymization from the applications deployed into SOA and implement them as a XML mediation by themselves. However, it is also difficult even for the developers, who know the technology generally used in the mediation such as XSLT [13], to implement the functions correctly without using any tool. Some technologies [4, 5, 6] have been proposed to support development of the functions by defining a conceptual data model and mapping the model to the implementation-specific data representation. But [4, 5] do not provide the general data model both for data validation and privacy anonymization and [6] does not provide the mappings between data model and real environment.

In this paper, we propose a uniform rule-based approach to realize the functions as a XML mediation separated from the applications, which makes it easy to maintain consistency of the functions when they are updated in accord with the changes to the specifications. Therefore, application developers can readily utilize it with many applications in SOA without additional modification to the applications. Our approach allows the developers to define a set of rules that consist of two components: constraint conditions in a conceptual data notation in the XML message and actions performed only when the conditions are satisfied. In order to make the rules independent from both the implementation-specific data notation and the industry-specific knowledge, we automatically transform the rules into the implementation-specific data representation, such as XACML [9], Schematron [11], or XSLT, using two more factors; one is the data mappings from the data notation in the rules to the concrete data representation in the implementation and the other is the implied data relationships hidden in the rules. It is very important to take into consideration a general way to import the implied knowledge because it often depends on the industry-specific data structure and it is usually given outside of the mediation system.

The rest of the paper is organized as follows. We first explain our motivation behind our research in Section 2, and then introduce some related work in Section 3. We describe the definition of our rules in Section 4 and present the framework of rule-based XML mediation in Section 5. Finally, we conclude the paper in Section 6.

2. Motivation

In general, the industry-specific specifications are too complicated for a developer to understand all of the data constraints and security constraints in the specifications. For example, let us assume healthcare industry. Even if the developer can understand the constraints in the Health Level 7 (HL7) [2] standard and develop some applications including the logics to validate the constraints, it is difficult to update the applications separately whenever the schema of the XML messages is continually changed

HL7 version 2.x supports only hospital workflow. Since early versions of HL7 2.x adopt a binary (non-XML) format, legacy applications still depend on this format. Also, the format is currently supported by most of major medical information systems vendors in the US for backword compatibility. However, when we want to migrate the existing applications over to a system in SOA, the application developers need to provide a mediation to convert v2.x binary format to a XML format for (possibly all of) the HL7-related applications and vice versa. The HL7 version 3 standard defines the XML message format but it does not specify all correspondence relationships between the data in the binary format and the field in the XML format completely although it supports any and all healthcare workflows. It would cause the interoperability problem.

To reduce the risk hidden in the data format conversion and make it easy to update and maintain the logics in accord with the changes to the schema of the XML messages, it is a good approach to separate the logic for data validation and for privacy anonymization from the application logic. This is why we propose a simple and uniform approach to realize XML mediation.

3. Related work

In the area of database applications, there is a large amount of work in semantic data modeling based on Entity-Relationship approach. For example, [4] defines a conceptual data model that extends the notion of the standard ER model to capture security constraints for the database application. Each security constraint is expressed in the security constraint language and graphically mapped into an ER model. It also gives the taxonomy of security semantics. Based on the taxonomy, it can detect conflicting constraints and let the system designer know what conflicts exist. [5] uses a conceptual data model as a Platform Independent Model (PIM) and the concrete data model as a Platform Specific Model (PSM). The PIM is represented using an extended UML class diagram that includes the security aspects. It proposes a method to map from the PIM with secure data (security constraints on the conceptual data model) to the PSM using secure data (a secure data representation in the concrete implementation). As with [4], the basic concept is the same, but there is a difference in the descriptive power of the model. As typified by [4, 5], the approaches for the database applications are basically designed to describe only data security constraints satisfied in the data model. For example, the constraints include which data field is confidential when an instance of the data satisfies a condition. Because the model does not consider data validation constraints such as data consistency, the application developers are responsible for checking the data validation by themselves.

In the area of natural language processing, there is a technology that transforms the privacy rules written in a natural language into a XACML policy as defined in the SPARCLE system [6]. The approach in [6] is a similarity to our approach in the generation of XACML policies, but this work just uses the XAML policy to represent the conceptual model that we mentioned earlier in the paper and provide no concrete data mapping between the model and real environments. The application developers need to define the data mapping by themselves in their each application and implement the mechanism to convert the XACML policy to the application-specific representation of the logics.

Model-based XML mediation itself is already proposed in [7]. But the model in [7] is just used to fill the semantic gap between the interfaces of two services and make it possible to exchange XML message between the services. Since the model is automatically converted into XSLT, it can be used as the XML mediation. Although we have the different purpose from [7], our approach uses the idea of model-based XML mediation.

4. Definition of the rules

We propose a simple and uniform approach to realize XML mediation, such as data validation or privacy anonymization, based on a set of rules that consists of constraint conditions in a conceptual data notation in the XML message and actions performed only when the conditions are satisfied. We define the details of the rules in this section. There are some similar technologies [4, 5, 6] to define security and privacy constraints, but the constraints handled in those technologies can be regarded as one aspect of our definition.

The definition of rules is a set of rules represented as *R+*. Each element *R* in *R+* consists of two parts: A condition *C* and an action *A*. *A* is performed only when the corresponding *C* is satisfied. *C* is a Boolean condition *BC* or a Boolean expression of two *C*s associated with a logical operator like *AND*, *OR*, or *NOT*. *A* is a simple operation, such as rejecting the XML message or anonymizing a specific element in an XML message. *BC* is one of three types of evaluation formulas, F_1, F_2, or F_3. F_1 is a function that requires only one parameter *O*. F_1 includes the 'is null' function that returns true when the parameter is null. F_2 is a function that requires two parameters as a combination of *O* or *MVF* defined below. Binary operators such as '=' are examples of F_2. F_3 is a function that requires two parameters, *O* and O_{set}. An example of F_3 is the membership function that checks *O* is included in O_{set}. *MVF* is a multivariable function represented as $MVF(O_1, O_2, ..., O_n)$, where $O_i \in O$ $(1 \leq i \leq n)$, and it returns a value calculated from some *O*s defined below. For example, *MVF* includes the function to calculate a person's age using his birthday. *O* is a conceptual notation for three types of data: constants *OC*, parts of the XML message *OI*, or data outside of the message *OO*. We use symbols like *OI(x)* to identify the same object in a rule. Our definition of the rules allows us to flexibly add other definitions to F_1, F_2, F_3, or *A* if needed. For example, a user could add a function to sum two *O*s for an F_3 function. We can also represent *C* graphically as a tree hierarchy. Our definition of the rules is summarized as follows:.

Definition of rules: *R+*

R: $C \rightarrow A$

C: *BC* | *(C AND C)* | *(C OR C)* | *(NOT C)*

BC: $F_1(O)$ | $F_2(O, O)$ | $F_2(MVF, O)$ | $F_2(O, MVF)$ | $F_2(MVF, MVF)$ | $F_3(O, Oset)$

F_1: Boolean function composed of a term; is null | …

F_2: Conditional operation composed of two terms; $=$ | $<$ | $>$ | $\leq$ | $\geq$ | $\neq$ | …

```
<s:Envelope>
 <s:Body>
  <message type="response">
   <patient id="001">
    <attendingDoctor id="002" />
    <status>not good</status>
    <course>…</course>
   </patient>
   <persons>
    <person id="001">
     <name>Dice-K</name>
     <address>Boston, MA</address>
     <birthday>20000101</birthday>
     <parent id="003" />
    </person>
    <person id="002">
     <name>Hideki</name>
     <profession>Doctor</profession>
    </person>
    <person id="003">
     <name>Ichiro</name>
    </person>
   </persons>
  </message>
 </s:Body>
</s:Envelope>
```

Figure 1. An example of the XML message

F_3: Function that checks a data is included in a set of another data

MVF: Multivariable function; $MVF(O_1, O_2, ..., O_n)$

O: Conceptual notation of three types of single data; *OC* | *OI* | *OO*

OC: Constants

OI: Conceptual notation of parts of the message

OO: Conceptual notation for data outside of the message

O_{set}: Conceptual notation of a set of data

A: Simple operation; Reject a message *Rm* | Anonymize a conceptual notation *D(OI)* | …

In this paper, we classify the definition of rules into two categories: privacy constraint rules that use *D(OI)* as *A* and data constraint rules that uses *Rm* as *A*. This classification is used in Section 5, although we will see in the following discussion that our approach is applicable to other categories. In the next section, we show some examples of rules written in a natural language and their corresponding formal representations following the above definition of rules. We are not focusing on the way to transform a rule written in a natural language into a formal one. We assume that the transformation is currently done manually. Our future work will include ways to apply the existing technologies [6, 8] in this area to the transformations.

We use the XML message shown in Figure 1 to show a concrete application of the rules. In Figure 1, the patient element includes detailed information about a patient, like status and course of treatment. The 'persons' element includes a set of person elements. A person element has detailed information about each person such as name and address. There are some restrictions on external reference in the example. For example, the identifier of a patient may need to be registered in a database of a patient information system. The same holds for the patient's parent and the patient's attending doctor. These restrictions are natural in real applications and should be provided in advance based on the data model.

4.1. Simple constraint condition (SCC)

Here are some examples of the rules that depend on the evaluation results of simple conditions.

(r1) Rule written in natural language: "The message has to be rejected when the identifier of a patient is null."
Data constraint rule: $F_{1\text{-}a}(Patient) \rightarrow Rm$ where $F_{1\text{-}a}$ is the 'is null' function and *Patient* is an *O* that identifies a patient.
Concrete application: *Patient* is the attribute value specified with the XPath expression '*//patient/@id*' in Figure 1. In this example, the message is not rejected since the patient's identifier is '001' (not null).

(r2) Rule written in natural language: "The patient's treatment has to be anonymized when the client is not the patient."
Privacy constraint rule: $F_{2\text{-}a}(Client,\ Patient(x)) \rightarrow D(Patient(x).Treatment)$, where $F_{2\text{-}a}$ is the operator '$\neq$', *Client* is an O that identifies a client, and *Patient(x).Treatment* is an *O* that identifies a patient's treatment.
Concrete application: *Client* is the identifier extracted from the result of client authentication done outside of the message. *Patient(x).Treatment* consists of two texts specified with XPath expressions, '*//patient/status/text()*' and '*//patient/course/text()*' in Figure 1. In this example, if the client's identifier is not '001', the patient's status and course of treatment will be anonymized.

(r3) Rule in natural language: "The patient's address also has to be anonymized when the client is not the patient."
Privacy constraint rule: $F_{2\text{-}a}(Client,\ Patient(x)) \rightarrow D(Patient(x).Address)$, where *Patient(x).Address* is an *OI* that identifies the patient's address.
Concrete application: *Patient(x).Address* is the text specified with the XPath expression '*//person/address/text()*' in Figure 1. The (r3) is almost same as (r2) at the level of the rule written in natural language, but it requires an implicit relationship in order to apply the rule to the concrete implementation. In this example, we need to identify who the patient is by using the patient's identifier. To implement this, we need an additional data relationship, $F_{2\text{-}b}(Patient(x),\ Person)$, where $F_{2\text{-}b}$ is the operator '=' and *Person* is the attribute value specified with the XPath expression '*//person/@id*' in Figure 1. If the client's identifier is not '001', the text in the patient's address will be anonymized.

(r4) Rule written in natural language: "The message has to be rejected when the profession of the patient's attending doctor is not 'Doctor'."

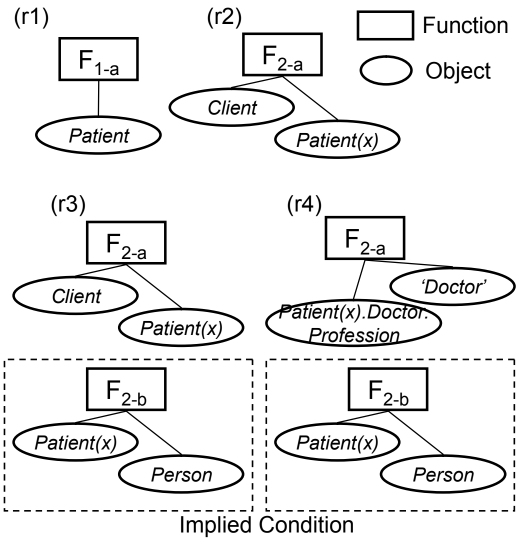

Figure 2. Graphical representation of SCCs

Data constraint rule: $F_{2\text{-}a}$ *(Patient(x).Doctor.Profession, 'Doctor')* $\rightarrow Rm$, where *Patient(x).Doctor.Profession* is an *O* that identifies the profession of the patient's attending doctor.
Concrete application: *Patient(x).Doctor.Profession* is the text specified with the XPath expression '*//person/profession*' in Figure 1. In this example, we need an additional data relationship $F_{2\text{-}b}(Patient(x),\ Person)$ as well as (r3). In this example, the message is not rejected since the profession of the patient's attending doctor is 'Doctor'.

Figure 2 shows a graphical representation of these SCCs.

4.2. Complex constraint condition (CCC)

Here are some examples of rules that depend on evaluation results of combinations of more than one condition.

(r5) Rule written in natural language: "The message has to be rejected if there is a person whose parent is not in the set of persons."
Data constraint rule: $(NOT\ F_{3\text{-}a}(Person.Parent,\ PersonSet)) \rightarrow Rm$, where $F_{3\text{-}a}$ is a new function 'exist in' that returns true if the first parameter exist in the second parameter, *Person.Parent* is an *O* that identifies a parent of the person, and PersonSet is an O_{set} that identifies a set of persons.
Concrete application: *Person.Parent* is the attribute value specified with the XPath expression '*//person/parent/@id*' and *PersonSet* is a set of attribute values specified with the XPath expression '*//person/@id*' in Figure 1. In this example, the identifier of each parent

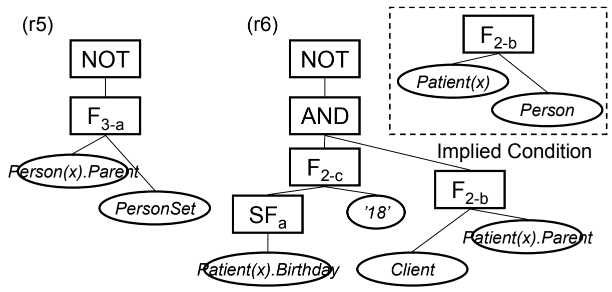

Figure 3. Graphical representation of CCCs

would be checked if it is included in a set of the identifiers of persons because each person has his parent's identifier. Since the identifier '003' of a person's parent matches with another person's identifier, the message may not be rejected.

(r6) Rule written in natural language: "The patient's treatment has to be anonymized when the client is the patient's parent but the patient is over 18."
Privacy constraint rule: *(NOT ($F_{2\text{-}c}$(SF_a(Patient(x).Birthday), 18) AND $F_{2\text{-}b}$(Client, Patient(x).Parent)))* → *D(Patient(x).Treatment)*, where $F_{2\text{-}c}$ is '<' function, SF_a is a new function that calculates the age of a patient, *Patient(x).Birthday* is an *O* that identifies the patient's birthday, and *Patient(x).Parent* is an *O* that identifies the patient's parent.
Concrete application: *Patient(x).Birthday* is the text specified with the XPath expression '*//person/birthday/text()*' and *Patient(x).Parent* is the attribute value specified with the XPath expression '*//person/parent/@id*' in Figure 1. In this example, we need the data relationship *$F_{2\text{-}b}$(Patient(x), Person)* as well as (r3). If the client's identifier is '003', he can see the patient's address since it equals the identifier of the patient' s parent and the patient is under 18.

Figure 3 shows a graphical representation of these complex constraint conditions.

5. Framework of rule-based XML mediation

In this section, we define more implementation-specific data mappings to realize XML mediation based on the rules defined in the previous section.

5.1. Implementation-specific data mapping

We define data mappings from the conceptual data notation to an implementation-specific data representation as a set of pairs of key and value. The key is the conceptual data notation of the rules defined in the previous section, such as Client, Patient, or Person. The value is an implementation-specific data representation using XPath expression such as '//patient/@id' or some function that outputs a value such as an extraction of the client identifier from the result of client authentication done outside of the message. Table 1 shows the pairs of key and value used for the examples in Section 4. When we transform the conceptual data notation into an implementation-specific data representation, we will also require additional data relationships that are hidden in the rules. The (r3) used in Subsection 4.1 is a good example. The XML message used in Section 4 includes more than one person element. If a different person has a different address, then the relationship between the *Patient* (*//patient/@id*) and the *PatientAddress* (*//person/address/text()*) are underspecified. Therefore, we need to define such implied data relationships to make them uniquely-determined. We assume that these implicit data relationships should be given in advance based on the data model but we need to consider how to derive the relationships from the data model in the future. Table 2 shows the implied data relationships used in the examples in Section 4.

There are various technologies to implement XML mediation. We can currently transform our rules to the descriptions that are processable by the following technologies:

- For privacy-constraint rules in isolation, we can use the existing XACML implementation [10] to process the policy in the XML mediation after we transform the rules to a policy defined in the XACML.
- For data-constraint rules in isolation, we can use the existing Schematron implementation [12] to convert

Table 1. An example of implementation specific data mapping

Key (data notation)	Value (semantics)
Patient	XPath: *//patient/@id*
PatientTreatment	XPath: *//patient/status/text()* and XPath: *//patient/course/text()*
PatientAddress	XPath: *//person/address/text()*
PatientBirthday	XPath: *//person/birthday/text()*
PatientParent	XPath: *//person/parent/@id*
PatientAttendingDoctorProfession	XPath: *//person/profession/text()*
Client	Function: Extraction of the client identifier from the result of client authentication outside the message
Person / PersonSet	XPath: *//person/@id*

Table 2. An example of an implied data relationship

Key (data notation)	Value (semantics)
$F_{2\text{-}b}$(Patient, Person)	Returns true if the patient's identifier is equal to the person's identifier

the Schematron rules to XSLT for the XML mediation after we transform the rules to Schematron rules.

- Both for privacy-constraint rules and for data-constraint rules, we can use the existing XSLT processor [14, 15] directly in the XML mediation after we transform the rules to XSLT processing rules.

5.2. Rule transformation to XACML

When we transform the privacy-constraint rules defined in Section 4 to XACML policies, we do not necessarily need the data mappings and implied data relationships defined in Subsection 5.1 because we can describe the XACML policies in a way that is independent of any XML data representation. However, in this subsection we show how to transform the rules to the XACML policies using data mappings and implied data relationships.

The condition related to the client is mapped into the Subject. Action *D(OI)* is mapped into the Rule's effect and the Action of XACML. The object *D(OI)* is mapped into the Resource. The rest of the conditions are mapped into the Condition. When the Condition is constructed, the data mappings and implied data relationships are also used. Figure 4 shows a snippet of the XACML policy transformed from (r6) as described in Subsection 4.2, showing only the key points. In Figure 4, the XPath expression '*//person/parent/@id*' is mapped to the Subject by using $F_{2\text{-}b}$*(Client, Patient(x).Parent)*. The Rule's effect is set to 'Deny', The Action is set to 'read', and both of the XPath expressions '*//patient/status/text()*' and '*//patient/course/text()*' are mapped into the Resource by using *D(Patient(x).Treatment)*. The condition $F_{2\text{-}c}$*(*SF_a*(Patient(x).Birthday), 18)* and the implied condition $F_{2\text{-}b}$*(Patient(x), Person)* are mapped into the Condition. The calculation of the patient's age is based on the semantics defined in the data mappings.

5.3. Rule transformation to Schematron

A Schematron description is generally used only for data validation, although it is similar to XSLT processing rules. When we transform the rules defined in Section 4 to Schematron rules, we can map the data constraint conditions to Schematron rules based on the hierarchy of the conditions and reject the XML message. Figure 5 shows a snippet of the Schematron description transformed from (r1) as described in Subsection 4.1, showing only the key points. In Figure 5, the XPath '*//patient/@id*' is embedded into the XPath evaluation. If the XPath evaluation is false, then the message is rejected.

```
<Policy xmlns="xacml:1.0:policy"
 PolicyId="sample-rules“
 RuleCombiningAlgId="rule-combining-algorithm:deny-overrides">
  <Rule RuleId= "sample-rules:rule1" Effect="Deny">
   <Target>
    <Subjects>
     <Subject>
      <SubjectMatch MatchId="function:string-xpath-node-match">
       <AttributeValue DataType="#string">
        /s:Envelope/s:Body/message/persons/person/parent/@id
       </AttributeValue>
      </SubjectMatch>
     </Subject>
    </Subjects>
    <Resources>
     <Resource>
      <ResourceMatch MatchId="function:xpath-node-match">
       <AttributeValue DataType="#string">
        /s:Envelope/s:Body/message/patient/status/text()
       </AttributeValue>
      </ResourceMatch>
     </Resource>
     <Resource>
      <ResourceMatch MatchId="function:xpath-node-match">
       <AttributeValue DataType="#string">
        /s:Envelope/s:Body/message/patient/course/text()
       </AttributeValue>
      </ResourceMatch>
     </Resource>
    </Resources>
    <Actions>
     <Action>
      <ActionMatch MatchId="function:string-equal">
       <AttributeValue DataType="#string">read</AttributeValue>
      </ActionMatch>
     </Action>
    </Actions>
   </Target>
   <Condition FunctionId="function:and">
    <Apply FunctionId="function:integer-equal">
     <AttributeSelector RequestContextPath=
"/s:Envelope/s:Body/message/patient/@id“
      DataType="#integer"/>
     <AttributeSelector RequestContextPath=
"/s:Envelope/s:Body/message/persons/person/@id"
      DataType="#integer"/>
    </Apply>
    <Apply FunctionId="function:date-greater-than-or-equal">
     <Apply FunctionId="function:date-one-and-only">
      <EnvironmentAttributeDesignator
       AttributeId="environment:current-date" DataType="#date"/>
     </Apply>
     <Apply FunctionId="function:date-add-yearMonthDuration">
      <Apply FunctionId="function:date-one-and-only">
       <AttributeSelector RequestContextPath=
"/s:Envelope/s:Body/message/persons/person/birthday/text()"
        DataType="#date"/>
      </Apply>
      <AttributeValue DataType="#date">18-00-00</AttributeValue>
     </Apply>
    </Apply>
   </Condition>
  </Rule>
</Policy>
```

Figure 4. The XACML policy transformed from (r6) in Subsection 4.2.

```
<schema>
 <pattern name="rule1">
  <rule context="/">
   <assert test=“string-length(/s:Envelope/s:Body/message/patient/@id) &gt; 0">
   </assert>
  </rule>
 </pattern>
</schema>
```

Figure 5. The Schematron description transformed from (r1) in Subsection 4.1.

```
<!-- Rule 1 -->
<xsl:template match="//message/patient">
  <xsl:variable name="_patientid_" select="@id"/>
  <xsl:choose>
    <xsl:when test="string-length($_patientid_) &gt; 0"/>
    <xsl:otherwise>Violation in Rule 1</xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- Rule 2 -->
<xsl:template match="//message/patient/status">
  <xsl:variable name="_clientid_" select="001"/>
  <xsl:variable name="_patientid_" select="//message/patient/@id"/>
  <xsl:choose>
    <xsl:when test="$_clientid_ != $_patientid_">
      <xsl:copy>Deidentified by Rule 2</xsl:copy>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy>…</xsl:copy>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
<xsl:template match="//message/patient/course">
  …
</xsl:template-->

<!-- Rule 3 -->
<xsl:template match="//message/persons/person/address">
  <xsl:variable name="_clientid_" select="001"/>
  <xsl:variable name="_patientid_" select="//message/patient/@id"/>
  <xsl:variable name="_personid_" select="../@id"/>
  <xsl:choose>
    <xsl:when test="
($_personid_ != $_patientid_) or ($_clientid_ != $_patientid_)
      ">
      <xsl:copy>Deidentified by Rule 3</xsl:copy>
    </xsl:when>
    <xsl:otherwise>
      <xsl:copy>…</xsl:copy>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<!-- Rule 4 -->
<xsl:template match="//message/patient/attendingDoctor">
  <xsl:variable name="_doctorid_" select="@id"/>
  <xsl:choose>
    <xsl:when test="
//message/persons/person[@id=$_doctorid_]/profession/text() =
'Doctor'
      "/>
    <xsl:otherwise>Violation in Rule 4</xsl:otherwise>
  </xsl:choose>
  <xsl:copy>…</xsl:copy>
</xsl:template>
```

Figure 6. The XSLT processing rules transformed from (r1) – (r4) in Subsection 4.1.

5.4. Rule transformation to XSLT

We can handle both data-constraint rules and privacy-constraint rules as defined in Section 4 using the same XSLT implementation if we transform the rules to XSLT processing rules. Figure 6 shows the XSLT processing rules transformed from (r1)-(r4) as described in Subsection 4.1, again simplified for illustrative purposes. When we transform the simple constraint conditions described in Subsection 4.1, we can apply simple mapping rules as follows:

- Define a context root for a template. For example, in (r1) we can use '*//patient*' as the context root.
- Store the necessary data into the variables. For example, the patient's identifier is stored in the variable '*_patientid_*' and the client's identifier is stored in the variable '*_clientid_*' in (r2).
- Transform the condition into the XPath evaluation. For example, the XPath evaluation '*($_personid_ != $\_patientid\_) or ($_clientid_ != $_patientid_)*' is generated by $F_{2\text{-}a}(Client, Patient)$ *and* $F_{2\text{-}b}(Patient, Person)$ in (r3).
- Decide the behaviors based on the actions. If a message is rejected, an exception will be thrown. When a part of the message is determined to be anonymized, that part may be removed from the original message. For example, the message is rejected with an exception when the condition is false in (r4).

5.5. Rule transformation to XSLT

When we transform the complex constraint conditions described in Subsection 4.2, we sometimes need a template for a rule transformation, especially for XSLT processing rules. Figure 7 shows the XSLT processing rules transformed from (r5)- (r6). Again the figure has been simplified here.

We define a new function $F_{2\text{-}c}(Person(x).Parent, Person)$ in (r5) in Subsection 4.2. The following template realizes this function as an XSLT processing rule. A sample of the XSLT rule is shown in Figure 7.

a) Whenever an element including the *Person(x).Parent* is found,
   i. Extract the *Person(x).Parent* and store it in the variable '*_parentid_*'.
   ii. Check if the Person includes '*$_parentid_*' using the function 'count'. If it is not included (i.e. the count is zero), the message is rejected.

We also define a new function $SF_a(Patient(x).Birthday)$ in (r6) in Subsection 4.3. For this function, the template is given by the following. A sample of the XSLT rule is shown in Figure 7.

b) When an element including the *Patient(x).Birthday* is found,
   i. Extract the *Patient(x).Birthday* and store it in the variable '*_sd_*'.
   ii. Extract the current date and store it in a variable like '*_cd_*'.
   iii. Calculate the patient's age using '*$\_sd\_*' and '*$_cd_*' (being careful about the date formats).

```
 <!-- Rule 5 -->
 <xsl:template match="//message/persons/person/parent">
   <xsl:variable name="_parentid_" select="@id"/>
   <xsl:if test="string-length($_parentid_) &gt; 0">
     <xsl:choose>
       <xsl:when test="
count(//message/persons/person[@id = $_parentid_]) &gt; 0
       "/>
       <xsl:otherwise>Violation in Rule 5</xsl:otherwise>
     </xsl:choose>
   </xsl:if>
 </xsl:template>

 <!-- Rule 6 -->
 <!--xsl:template match="//message/patient/status">
   <xsl:variable name="_clientid_" select="001"/>
   <xsl:variable name="_patientid_" select="//message/patient/@id"/>
   <xsl:variable name="_parentid_" select="
//message/persons/person[@id=$_patientid_]/parent/@id
   "/>
   <xsl:variable name="_cd_" select="date:date-time()" />
<xsl:variable name="_cdy_" select="substring($_cd_, 1, 4)"/>
   <xsl:variable name="_sd_" select="
//message/persons/person[@id=$_patientid_]/birthday/text()
   "/>
   <xsl:variable name="_sdy_" select="substring($_sd_, 1, 4)"/>
   <xsl:variable name="_svy_" select="18"/>
   <xsl:choose>
     <xsl:when test="
not (($_patientid_ = $_clientid_) and ($_cdy_ &lt; $_sdy_ + $_svy_))
       ">
       <xsl:copy>Deidentified by Rule 6</xsl:copy>
     </xsl:when>
     <xsl:otherwise>
       <xsl:copy>…</xsl:copy>
     </xsl:otherwise>
   </xsl:choose>
 </xsl:template-->
 <xsl:template match="//message/patient/course">
   …
 </xsl:template>
```

Figure 7. The XSLT processing rules transformed from (r5)-(r6) in Subsection 4.2..

Since we considered only two cases in this paper, we need more investigation and discussion in order to formalize these templates and to include them into our definitions. This is part of our future work. Also, we may find conflicts in a set of rules. For example, both (r2) and (r6) define *D(Patient(x).Treatment)* as the action. To resolve conflicts, the system checks all the conceptual notations used in the actions and, for each notation that is used more than one action. Then, the system can ask the rule designer which one is preserved or in which order those rules are applied or may be able to merges the rules.

6. Concluding Remarks

In this paper, we have presented a uniform rule-based approach to realize the functions as a XML mediation for data validation and privacy anonymization, which makes it easy to maintain consistency of the functions when they are updated in accord with the changes to the specifications. Therefore, application developers can readily utilize it with many applications in SOA without additional modification to the applications. Our approach allows the developers to define a set of rules that consists of constraint conditions in a conceptual data notation in the XML message and actions performed only when the conditions are satisfied. In order to make the rules independent from both the implementation-specific data notation and the industry-specific knowledge, we automatically transform the rules into the implementation-specific data representation using two more factors; data mappings from the data notation in the rules to the concrete data representation in the implementation and the implied data relationships hidden in the rules.

Our future work will include application of existing natural language processing techniques to generate the rules, validation of our definitions for rules, and formalization of the template descriptions required for new functions defined in the rules.

References

[1] Simple Object Access Protocol (SOAP) Version 1.2, http://www.w3.org/TR/soap12/
[2] Healthcare Level Seven (HL7), http://www.hl7.org/
[3] Extensible Business Reporting Language (XBRL), http://www.xbrl.org/
[4] G. Pernul, A. M. Tjoa and W. Winiwarter, "Modeling Data Secrecy and Integrity", Data & Knowledge Engineering, Vol. 26, pp. 291-308, 1998.
[5] B. Vela, E. F. Medina, E. Marcos, and M. Piattini, "Model driven development of secure XML databases", ACM SIGMOD Record, Vol.35 No.3, pp.22-27, 2006.
[6] C. A. Brodie, C. Karat, and J. Karat, "An Empirical Study of Natural Language Parsing of Privacy Policy Rules Using the SPARCLE Policy, " Proceedings of the second symposium on Usable privacy and security (SOUPS '06), pp. 8-19, 2006
[7] A. Tolk, "XML Mediation Services Utilizing Model-Based Data Management," Proceedings of IEEE Winter Simulation Conference, IEEE CS Press, pp. 1476–1484, 2004.
[8] R. J. Kate, Y. W. Wong, and R. J. Mooney, "Learning to Transform Natural to Formal Languages," in Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05), pp. 1062--1068, 2005.
[9] Core Specification: eXtensible Access Control Markup Language (XACML) Version 1.0, http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf
[10] Sun's XACML implementation, http://sunxacml.sourceforge.net/
[11] ISO Schematron, http://www.schematron.com/spec.html
[12] Schematron implementation, http://xml.ascc.net/schematron/1.5/basic1-5/schematron-basic.html
[13] XSL Transformations (XSLT) Version 1.0, http://www.w3.org/TR/xslt
[14] SAXON – The XSLT and XQuery processor, http://saxon.sourceforge.net/
[15] Xalan-Java, http://xml.apache.org/xalan-j/