Representing and Reasoning on XForms Document

Peng Yew Cheow **Guido Governatori**

School of Information Technology and Electrical Engineering
The University of Queensland
Brisbane, Queensland, QLD 4072, Australia
Email: s4019325@student.uq.edu.au, guido@itee.uq.edu.au

Abstract

Forms are the most common way to interface users and Web-based applications. Traditional forms cannot provide the functionality needed to fulfil the requirements of complex applications. As such, there is a need for a more advanced format of forms to support Web-based application. We argued that XForms easily fit into this criterion of forms. In addition, we observed that there is a need for a tool to reason about the forms with respect to user needs and application requirements. We propose to use Description Logic $\mathcal{ALCOQI}$ to reason about forms generated by XForms.

1 Introduction

Web-based applications have been increasing in size and complexity ever since the sudden explosion of E-commerce in the 1990s. It has been observed that most of these applications are database-driven and exploit a common interface called forms for computer users to interact with applications. As such, these are the two important roles that determine the successful deployment of Web-based applications, namely *databases* and *forms*. The usage of the database is to facilitate the storage of application data via schemas. A schema describes the structure of the data stored in a database and often applications require schemas from various data sources to perform their intended tasks. This may pose a problem when data from various sources must be integrated as each schema has its own independent rules of describing the structure of the data. However, it is possible for the schemas of the various data sources to be conceived of as ontologies; in this way both conceptual and logical aspect of these schemas be represented in a formal language such as Description Logic [2]. It is thus feasible for the language to integrate data from the ontologies and check the consistency of the integrated data.

Forms on the other hand are connected to databases for either gathering data from users, or accessing the data from a database, or both. Over the years, Web applications have however sparked better forms for users with richer interactions. Traditional forms, generated by using Hypertext Markup Language (HTML), may be quite a tedious chore to perform the above mentioned tasks due to their limitations such as heavy dependence on scripting languages, design of page flow in forms and etc [13, 1, 14]. *XForms* [10] on the other hand offer a powerful and versatile tool to interface the Web and databases. Although XForms are still in their embryonic state, there has been much excitement about them on the Web and many companies have started to develop forms based on XForms[1]. In this paper, XForms are used as the standard for Web forms. XForms are chosen as the standard mainly due to the fact that they use XML schema as part of the component in the model element. The schema plays a critical role as the schema provides information about the domain of the form, which is then used to represent the semantics of the form. Moreover it is believed that XForms are going to inherit the role of generating web forms from HTML in the near future, due to their promising advantages.

Although XForms could resolve much of the weakness in the current markup languages, there are still limitations in the forms used by current Web applications. There has been no improvement made on the forms in terms of user needs and requirements of Web applications. Different users have different needs and, at the same time, different Web applications may require different domains of the field of that application. Hence forms should have the ability to intelligently suit both computer users and the requirements of the application systems. But, one may wonder, why is there still a need to verify the validity and logical design of forms, especially when the application's designers have already performed software requirement specification (SRS) on the needs of their applications? According to our knowledge, system developers tend to focus more on the application's needs than users, which in some instances may not realize actual user's needs aspect and vice visa. Whereas our approach takes into consideration of both party's needs, users and application respectively, when generating forms. Thus solving some of the limitation of current Web forms.

The problems of user needs and system requirements can be categorized into form validity and logical design of forms. For example, one of the most problems of form validity is that some of the fields in the form may be either irrelevant or inappropriate for users, making the form not fillable. In some other instances, the correct input of a user in some of the fields may perhaps produce wrong data in the subsequent fields, therefore making the form invalid for the user. Forms that are not well formed or do not conform to the specifications of their markup languages may produce erroneous form interactions to the user. As such, the syntax and semantics of a form play a very crucial role in determining the validity of the form. The logic design of some forms may require users to go through a certain number of forms to carry out their intended goals. Very often there are many unnecessary steps. These steps could however be reduced by embedding one form onto another, without causing interruptions to the workflow of the forms.

In order to improve logical design and the validity of the form, factors such as reasoning mechanisms for embedding of forms, and syntactic and semantic aspects need to be taken into consideration. Currently there is no reasoning mechanism for forms, especially for the new emerging XForms. Therefore the main aim of this paper is to provide a formal reasoning mechanism for XForms.

[1] See for example Formfaces (http://www.formfaces.com), TrustForm System (http://trustform.comsquare.co.kr) and the W3C XForms web page www.w3c.org/MarkUpX/Forms/ for more examples.

This objective is achieved by representing XForms documents in Description Logic. According to our knowledge, there is no research in applying Description Logic to reason about the semantics of XForms; moreover the current approaches to XForms validity focus only on the syntactic aspects, that is whether an XForms document conforms to the syntax of XForms. As such, there is a need to explore the context of semantic validity. In addition, the proposed reasoning mechanism could be implemented into an existing DL system such as FaCT system [12] to reason with and about XForms documents. Extensible Stylesheet Language Transformations (XSLT) would be use to form a communication bridge between XForms documents and DL systems.

The semantics of an XForms document would be translated by XSLT into a data format that is interpretable by the DL system. This data would then be used by the DL system and using the proposed reasoning framework to reason about form validity and form embedding. The DL system would update the data format on addition or deletion of data. After the data have been reasoned/updated, the new data format is translated by XSLT back into the original format interpretable by the XForms document, and a new Web form would be generated based on the reasoned XForms document. Thus the use of an expressive formal language enables us reason with and about XForms and the proposed implementation implies that user needs can be effectively captured and analysed, hence leading to interaction improvements between the forms and their users.

The paper is organized as follows. In Section 2, XForms are introduced and the basic reasoning task on forms are defined. In Section 3, the Description Logic $\mathcal{ALCOQI}$ is presented. In Section 4, the mapping process of XForms document to Description Logic is developed. In Section 5, we outline how to reason on form. Finally, conclusions are drawn in Section 6.

2 XForms

XForms, an extension module for XHTML, are the latest effort by the World Wide Web Consortium (W3C) to replace HTML forms with a more advanced format that could resolve many of the drawbacks and limitations of current markup languages. They are designed to be flexible and to work with other standard XML languages. XForms are to be integrated with other markup languages, and not adopted as a free-standing document type. It is theoretically possible to attach XForms's functions to almost everything [14]. This is important as Web developers then do not have to learn a new language, but they can simply use XForms for integration with the existing markup language such as XHTML, Scalable Vector Graphics (SVG), XSL and VoiceXML. The main advantages of XForms [1] over HTML are as follows:

1. **Powerful actions, event model and validation rules**: XForms provide a wide range of client-side processing and reduce the number of round-trips to the server without the need for scripting languages.
2. **Clean separation of data, logic and presentation**: This implies ease in generating data-bound controls.
3. **Highly regular XML structure**: The regular XML structure makes it possible to build WYSIWYG user interface (UI) development environments.
4. **Abstract controls**: This type of controls enable abstract application design that gets translated to device specific rendering.

2.1 XForms Document Structure

The structure of an XForms-based document can be described by using the XForms document of Figure 1. The `<model>` element in the `<head>` tag described a form definition, which controls a set of "rules" of how each form should conform to. In addition, this `<model>` element is also been used as a container for elements defined in the entire XForms model. The `<model>` element consists of submission information, schema, instance, data bindings and event handlers.

The `<schema>` element enables developers to define constraints for the returned data and is possible to link to an external documents rather than defining the data's constraints in the form itself. In addition to that, the `<schema>` element contains all the elements used in the form. The `<instance>` element references initial instance data and all XForms controls refer back to this element to store information provided by the user. The `<instance>` element can also be used to pre-populate a form. The data bindings, event handlers and submission information form the logic components of the form, which are then used to define the behaviour of the form. Data binding enables the specification of the types of the data entries, which can be related to other elements of the form, and includes some other features such as calculation and determining the relevance of the corresponding element's field via XPath. Events handlers are part of form controls defining how a form should behave when certain actions are triggered. Finally a form would be incomplete without the definition of some way of communicating with the back-end server. The function of the `<submission>` element is to pass, when activated, the data structure and the data to the location specified for the processing. The `<body>` contains the actual form controls used to collect inputs from users, populating forms, and specifying the presentation of the form. Form controls are expressed through both atomic and compound controls. The former are used to populate a form, while the latter are used to organise and group atomic objects.

2.2 Reasoning on Forms

The basic types of reasoning we can perform on forms are semantic and syntactic validity, and form embedding. Semantic validity is the verification of whether the form can be filled, integrated or if the form is consistent, while syntactic validity is the verification of whether the form conforms to its specifications. Form fillability is concerned with the relevance of the field's data with respect to its scheme constraint. Form integration, which very often requires the use of ontology servers, ensures that the form contains the essential and relevant fields needed for user input. Consistency of the form is to identify whether the field's data of the form correctly interpret its corresponding fields' data. Form embedding determines whether one form could be embedded in another form. Addition or deletion of fields in the forms will be based on these factors.

It is important to highlight at this point that the paper will be focusing only on the semantic validity and embedding of forms. Syntactic validity is not taken into consideration as form conforming to its specification could be easily verified by using commercial tools available in the Internet.

3 Description Logic $\mathcal{ALCOQI}$

In this section we introduce the description logic that will be used in Section 4 to represent XForms documents. The basic building blocks of Description Logics are concepts and roles [3]. Concepts are denoted as classes, describing the common properties of a collection of individuals while roles are interpreted as binary relations between objects. Complex concepts are built from a set of atomic concepts and a set of atomic roles by applying concept and role constructors.

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:xforms="http://www.w3.org/2002/01/xfroms">
<head>
  <xforms:model>
    <xforms:schema><!-- Information on form validation -->
      <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <xsd:element name="amexcard">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="fullName" type="xsd:string"/>
              <xsd:element name="password" type="xsd:string"/>
              <xsd:element name="personalInfo" maxOccurs="1">
                <xsd:complexType>
                  <xsd:sequence>
                    <xsd:element name="name" type="nameType"/>
                    <xsd:element name="homeAddress" type="homeaddressType"/>
                    <xsd:element name="ownMobile" type="xsd:string"/>
                    <xsd:element name="mobileNumber" type="xsd:integer"/>
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
          ...
        </xsd:element>
        <xsd:complexType name="nameType">
          <xsd:sequence>
                    <xsd:element name="title"/>
                    <xsd:element name="firstName" type="xsd:string"/>
                    <xsd:element name="middleName" type="xsd:string"/>
                    <xsd:element name="lastName" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
        <xsd:complexType name="homeaddressType">
          <xsd:sequence>
                    <xsd:element name="homeaddr" type="xsd:string"/>
                    <xsd:element name="city" type="xsd:string"/>
                    <xsd:element name="state" type="xsd:string"/>
                    <xsd:element name="zip" type="xsd:integer"/>
          </xsd:sequence>
          <xsd:attribute name="suburb" type="xsd:string"/>
        </xsd:complexType>
        ...
      </xsd:schema>
    </xforms:schema>
    ...
    <xforms:instance xmlns=""> <!-- The initial and final XML instance document -->
    </xforms:instance>
    ...
    <xforms:submission method="..." action="..." />
    ...
    <!-- Event definitions -->
    ...
    <!-- Binding information -->
    <xforms:bind nodeset="my:amexcard/my:fullName" required="true()" type="xsd:string"/>
    <xforms:bind nodeset="my:amexcard/my:password" required="true()" type="xsd:string"/>
    <xforms:bind nodeset="my:amexcard/my:personalInfo/my:name/my:title" required="true()" type="xsd:string"/>
    <xforms:bind nodeset="my:amexcard/my:personalInfo/my:name/my:firstName" required="true()" type="xsd:string"/>
    <xforms:bind nodeset="my:amexcard/my:personalInfo/my:mobileNumber" relevant="../my:ownMobile =' Y'"/>
    ...
  </xforms:model>
</head>
<body>
  <!-- Actual forms controls reside in the body of the document. -->
  ...
  <xforms:input ref="my:amexcard/my:personalInfo/my:name/my:firstName">
    <hint>Enter your first name.</hint>
    <help>Enter your first name.</help>
  </xforms:input>
  ...
  <xforms:selectOne ref="my:amexcard/my:personalInfo/my:name/my:title" selectUI="radio">
    <xforms:choices>
      <xforms:itemset>
        <xforms:value>Mr Mrs Ms Dr</xforms:value>
        <hint>Select your title.</hint>
        <help>It's selected your title.</help>
      </xforms:itemset>
    </xforms:choices>
  </xforms:selectOne>
  ...
</body>
</html>
```

Figure 1: A partial XForms document used for an online credit card application.

Description Logic $\mathcal{ALCOQI}$ has a rich combination of constructors, beside the basic constructors, it includes also qualified number restrictions, inverse roles, nominals and inclusion assertions of a general form. These constructors enable $\mathcal{ALCOQI}$ to be powerful enough to provide a unified framework for object-oriented languages and semantic data models as illustrated in [9, 4] and can represent XML documents [6]. In this paper we show that this logic can be used to capture the semantics of the XForms document. Concept descriptions in Description Logic $\mathcal{ALCOQI}$, which define how concepts and roles are formed, are built according to the syntax rules as shown:

$$\begin{aligned} C, C' \rightarrow & A | \neg C | C \sqcap C' | C \sqcup C' \\ & | \forall R.C | \exists R.C | \exists^{\geq m} R.C | \exists^{\leq m} R.C | \{a_1, \ldots, a_n\} \\ R \rightarrow & P | P^- \end{aligned}$$

where, A and P, denote atomic concepts and atomic roles respectively, C and R denote arbitrary concepts and roles, m is a positive integer and $\{a_1, \ldots, a_n\}$ are individual names. The basic reasoning tasks to be performed on concept expressive are satisfiability and subsumption. The former is used to test whether a newly defined concept is contradictory while the later is used to deduce whether a concept is more general than another. The constructors used to form concept expressions are the basic set operators, namely complement ($\neg$), union ($\sqcup$) and intersection ($\sqcap$) that are denoted as negation, disjunction and conjunction respectively. For example the concept of "person without children" can be expressed as

$$(\mathsf{Male} \sqcup \mathsf{Female}) \sqcap \neg \mathsf{Parent}$$

where we use the union (disjunction) of the concepts Male and Female for the notion of person and we intersect it with the complement (negation) of the concept Parent.

In addition, $\mathcal{ALCOQI}$ admits inverse roles and several forms of quantification to denote the inverse of a given relation and the representation of relationships existing between the objects in two classes respectively. In particular $\mathcal{ALCOQI}$ allows value restiction ($\forall R.C$), existential quantification ($\exists R.C$) and number restriction ($\exists^{\leq m} R.C$).

For example the concept $\forall$workIn.Male denotes the set of organizations with only male employees, while $\exists$workIn.Male identifies the set of organizations with at least one male employee; finally

$$\exists^{\leq 3} \mathsf{workIn}.\mathsf{Male}$$

defines the concept of organization with less than 4 male employees. More formally it denotes the set of instances of the concept defined the above expression are connected through the role workIn only to no more than 3 instances of the concept Male.

$\mathcal{ALCOQI}$ uses the role constructor that is the inverse role to denote the inverse of a given relation as:

$$\exists^{\leq 3} \mathsf{workIn}^-$$

One can for example state that there are 3 people who are unemployed by using the above expression. The logic also allows us to use individual names (nominals) not only in the ABox but also in the description language. The basic constructor for nominals is the "set" (or *one-of*) constructor represented by the expression

$$\{a_1, \ldots, a_n\}$$

where $a_1, \ldots, a_n$ are individual names. With the nominals constructor, it is possible to define the concept of countries participating in the commonwealth countries as {SINGAPORE, AUSTRALIA, INDIA, MALAYSIA, UNITED KINGDOM}.

Concepts are interpreted as subsets of a domain and roles as binary relations over the domain. An interpretation $I = (\Delta^I, \cdot^I)$ over a set $\mathcal{A}$ of atomic concepts and a set $\mathcal{P}$ of atomic roles consists of a nonempty finite set Δ^I (the domain of I) and a function $\cdot^I$ (the interpretation function of I) that maps every atomic concept $A \in \mathcal{A}$ to a subset A^I of Δ^I (the set of instances of A) and every atomic role $P \in \mathcal{P}$ to a subset P^I of $\Delta^I \times \Delta^I$ (the set of instances of P). The interpretation function can be extended to arbitrary concepts and roles with the following inductive definition:

$$\begin{aligned} (\neg C)^I &= \Delta^I \setminus C^I \\ (C_1 \sqcap C_2)^I &= C_1^I \cap C_2^I \\ (C_1 \sqcup C_2)^I &= C_1^I \cup C_2^I \\ (\forall R.C)^I &= \{o \in \Delta^I | \forall o'.(o,o') \in R^I \rightarrow o' \in C^I\} \\ (\exists R.C)^I &= \{o \in \Delta^I | \exists o'.(o,o') \in R^I \wedge o' \in C^I\} \\ (\exists^{\geq n} R.C)^I &= \{o \in \Delta^I | \sharp\{o' | (o,o') \in R^I \wedge o' \in C^I\} \geq n\} \\ (\exists^{\leq n} R.C)^I &= \{o \in \Delta^I | \sharp\{o' | (o,o') \in R^I \wedge o' \in C^I\} \leq n\} \\ (P^-)^I &= \{(o,o') \in \Delta^I \times \Delta^I | (o',o) \in P^I\} \\ \{a_1, \ldots, a_n\}^I &= \{a_1^I, \ldots, a_n^I\} \end{aligned}$$

3.1 Knowledge Bases in $\mathcal{ALCOQI}$

Description Logic $\mathcal{ALCOQI}$ provides facilities for setting up knowledge base through the use of concept expressions and assertions about individuals. A knowledge base is composed of two components, namely Terminological Box (TBox) and Assertional Box (ABox). The TBox contains intensional knowledge in the form of a terminology and is constructed through definition that states general properties of concepts and roles as follows:

$$\begin{aligned} A &\sqsubseteq C && \text{(inclusion assertion)} \\ A &\equiv C && \text{(equality assertion)} \end{aligned}$$

where A is an atomic concept and C is an arbitrary $\mathcal{ALCOQI}$ concept expression. The first definition is usually interpreted as inclusion assertion, which means that only necessary conditions are used for classifying instances of the concept A. Equality assertion (logical equivalence) specifies both necessary and sufficient conditions for the instances of the class.

The ABox contains extensional knowledge, describing concrete situations through assertions about individuals. The assertions to be used in this paper are concept and role assertion respectively, which are specified as follows:

$$C(a), \qquad R(b,c)$$

where C and R denote concepts and roles respectively, and individuals as a, b, c; for example:

$$\mathsf{Student}(\mathit{PengYew}) \quad \mathsf{workIn}(\mathit{ITEE}, \mathit{Guido})$$

Lastly, the last constructor to be used is the "fills" role for a role R which is denoted as $R : a$ and the semantic of this constructor is given as:

$$(R : a)^I = \{d \in \Delta^I | (d, a^I) \in R^I\}$$

where $R : a$ denotes the set of those objects that have a as filler of the role R. Thus we can interpret the following expression

$$\mathsf{workIn} : \mathit{Guido}$$

as Guido is currently holding a job in an organization, and the expression denotes the organizations Guido works for. Notice that this constructor makes possible to express role

assertions through concept assertions, that is an interpretation satisfies $R(a,b)$ iff it satisfies $(\exists R.\{b\})(a)$.

The semantics of a knowledge based is specified through the notion of satisfaction of assertions as specified in [9]. Given a knowledge base, an interpretation satisfies the inclusion assertion $A \sqsubseteq C$ if $A^I \subset C^I$ and it satisfies the equality assertion $A \equiv C$ if $A^I = C^I$. In addition, one can consider an interpretation is a model of a knowledge base if it satisfies all terminological assertions in it.

3.2 Reasoning Tasks in $\mathcal{ALCOQI}$

The fundamental reasoning tasks that can be carried out on the intensional level of the knowledge base are knowledge base satisfiability, concept consistency and logical implication. The formal definition of these types of reasoning according to [9] are given below:

1. **Knowledge base satisfiability**: A knowledge base K is satisfiable if and only if it admits a model.
2. **Concept consistency**: A concept C is consistent in a knowledge base K if and only if K admits a model where the interpretation of C is not empty.
3. **Logical implication**: A concept C_1 implies a concept C_2, $C_1 \sqsubseteq C_2$ in a knowledge base K if and only if in all models of K the interpretation of C_1 is a subset of the interpretation of C_2.

Concept consistency and logical implication generalize concept satisfiability and concept subsumption when we consider a knowledge base. These notions enable an user to deduce implicit knowledge from the knowledge that is explicitly contained in the knowledge base.

3.3 Applying Description Logic to Reason About Forms

As we have alluded to in Section 2.2 the basic types of reasoning we can perform on forms are form validity and form embedding. Here we examine how these operations can be carried out in Description Logic

1. **Consistency of forms**: We can use logical implication to verify whether a particular field has been correctly interpreted according to its corresponding field's data.
2. **Embedding of forms**: Given a representation of two forms F_1 and F_2 we have that F_1 is embedded in F_2 if $F_2 \sqsubseteq F_1$.
3. **Form integration with ontologies**: Knowledge base satisfiability and concept consistency can be used to decide whether a form contain all the essential and relevant fields, according to given ontologies.
4. **Form fillability**: Concept consistency can be used to verify the relevancy of the field's data; then this data is used to check the consistency of the form.

4 Representing XForms in Description Logic

To correctly represent XForms documents in Description Logic we have to examine four components of an XForms document: the XML Schema and `<bind>` elements declared in the head element of the document, XPaths declared in the whole document, and form controls declared in the `<body>` element of the document. The Knowledge Base of an XForms document is derived from these components. The resulting Knowledge Base is then used to verify the validity and logical design of the form. The schema, binding elements, XPath sand, sometimes, the form controls are used to structure the TBox in the Knowledge Base of an XForms document while the ABox is

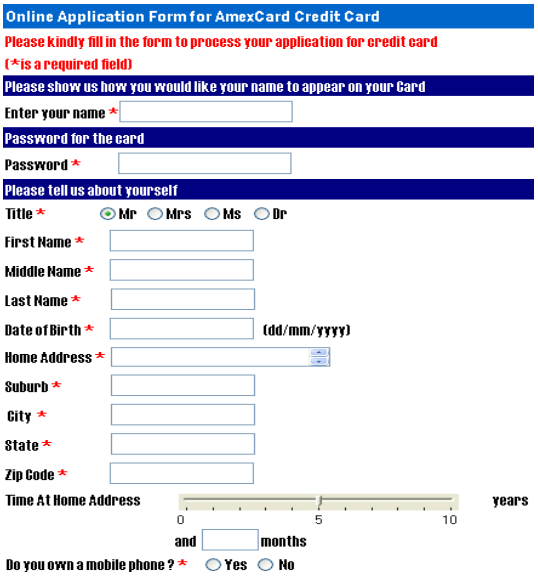

Figure 2: Graphical version of an XForms document used for an online credit card application.

structured via the form controls component. Figure 1 shows a partial XForms document for applying for a credit card and Figure 2 is the graphical version of that partial document. This example will be used in the rest of the paper to illustrate the mapping process from XForms to Description Logic.

4.1 Structuring the TBox

The tags of XML schema, attributes of binding element, XPaths and sometimes the form controls of an XForms document are used to structure the TBox in a Knowledge Base.

The schema of the document determines how the elements in it are represented as either atomic or complex concepts, and in case of complex concept it establishes the roles to be used and the type of quantification needed to define the complex concepts. Form controls are used only when pieces of information like cardinality constraints, are missing from the schema. The `<bind>` elements and XPath are used to verify or include additional information to the TBox.

Only element tags and their associated attributes are used to map the elements of the schema to concepts of Description Logic. Each element declared in the schema belongs to a domain of concepts, denoted as either atomic or complex concept.

Definition 1 A function called ConceptName is introduced to map each (complex) element in the schema to an assertion. This function receives an XForms document as input and aims to differentiate elements between atomic and complex concepts as output of that XForms document. $type(e)$ denotes the basic (built-in) type of the element e.

For each element e in an XForms document e is the concept corresponding to it in $\mathcal{ALCOQI}$. Then

If e's data type is built-in or simple then
  e $\sqsubseteq type(e)$
If e's data type is user-defined then
  If e's data type is complex then
    If e is of type `<xsd:sequence>` then
      e $\sqsubseteq \sqcap(RoleConcept(e') : e' \in$ `<xsd:sequence>`$)$
    If e is of type `<xsd:all>` then
      e $\sqsubseteq \sqcap(RoleConcept(e') : e' \in$ `<xsd:all>`$)$
    If e is of type `<xsd:choice>` then
      e $\sqsubseteq \sqcup(RoleConcept(e') : e' \in$ `<xsd:choice>`$)$
If e has attribute `ref=xpath` then

If `xpath` points to elements
$\mathsf{e} \sqsubseteq \sqcap(\mathsf{e}' \in \mathit{XPathE}(\mathit{xpath}))$
If `xpath` points to values
$\mathsf{e}(v)$ for all $v \in \mathit{XPathV}(\mathit{xpath})$

where $\mathit{XPathE}(\mathit{xpath})$ and $\mathit{XPathV}(\mathit{xpath})$ are, respectively, the set of elements and the set of values an XPath expression *xpath* refers to (see Definition 4).

The first element of the schema is defined as the type of the root element (document element), that is the element that specifies a document type. Each element has a name and may have a number of optional attributes of which only two attributes are relevant for the mapping, namely *data type* and *cardinality constraints*. The name of each element corresponds to the name of the concept. The data type of the element is used to verify whether the element itself is an atomic or a complex concept and cardinality constraints are used to establish the appropriate quantifiers and roles for complex concepts.

There are two methods for defining the data type of an element: built-in and user-defined. Element with built-in data type are mapped to atomic concepts. Elements with user-defined data type can be further broken down into simple and complex data types. The former data type is mapped to an atomic concept while the latter is mapped onto a complex concept. Notice that in some instances, an element of complex data type may use `<xsd:attribute>` tag instead of `<xsd:element>` tag to denote a child element. The former tag behaves exactly the same as the latter tag. As such, when encountered element with `<xsd:attribute>`, the attribute is treated as an element and follows exactly the same mapping process of a normal element.

Elements with simple data types are defined normally by using tags such as `<xsd:restriction>`, `<xsd:enumeration>` and others. These tags are, however, not relevant in mapping elements to concepts and therefore tags in simple data types are ignored. Complex data types are defined from existing data types by defining some attributes (if any) and by using `<xsd:sequence>`, `<xsd:all>` and `<xsd:choice>`. An elements containing a sequence of elements uses conjunction to construct the corresponding complex concept. Likewise for `<xsd:all>`, conjunction is also used in building complex concept.[2] Elements with `<xsd:choice>` are represented by the disjunction of the collection of elements, of which one will be chosen.

A complex concept is built by combining atomic and complex concepts with the function *RoleConcept* (Definition 2). This function establishes the types of quantifier and role for complex concept through cardinality constraints (*minOccurs*, *maxOccurs*) of the element and the element corresponding binding element information respectively.

The relationship between the elements in a complex concept is represented by a role, which is derived from the attributes of the binding element. The attributes of the binding element that are used to interpret role are *required* and *relevant*. The meaning of required is to indicate whether the domain of the corresponding element can be empty. The intuition behind the attribute relevant is that the existence of values for the element the attribute refers to depends on the values of other elements. The attribute `required` has two values, which can either be '`True()`' or '`False()`'. When the value is set to '`True()`', the role is represented by child-e in conjunction with existential quantification, and when set to '`False()`' the role is used with universal quantification.

[2]The order of the elements is relevant when an element is specified to be sequence of sub-elements. On the other hand, semantically, this does not change the meaning of the element itself. Technically there are no difficulties to represent such structure, However In this paper we are interested in the semantic meaning of the elements, thus we will ignore this difference.

Definition 2 The function *RoleConcept* establishs the types of quantifier and roles for complex concept. Each element in the schema is checked for cardinality constraints, where *n* and *m* are integers and then the element corresponding binding element's attribute for its role.

Let *e* be an element (attribute) in an XForms document.

If *e* has attribute `relevant="xpath='value'"`, then
$\mathit{RoleConcept}(e) = \mathit{Relevant}(e)$
otherwise
$\mathit{RoleConcept}(e) = \mathit{Concept}(e)$.

The function *Relevant* is thus defined

$$\mathit{Relevant}(e) = (\mathit{Role}(\mathit{xpath}){:}\{\mathit{value}\} \sqcap \mathit{Concept}(e)) \sqcap \neg \mathit{Role}(\mathit{xpath}){:}\{\mathit{value}\}$$

On the other hand the function *Concept* is thus defined

If *e* has attribute `required=True()` then
$\mathit{Concept}(e) = \exists \mathit{Role}(e).\mathsf{e}$
If *e* has attribute `required=False()` then
$\mathit{Concept}(e) = \forall \mathit{Role}(e).\mathsf{e}$
If *e* has $\mathit{minOccurs} = 0$ then
$\mathit{Concept}(e) = \forall \mathit{Role}(e).\mathsf{e}$
If *e* has $\mathit{minOccurs} = 1$ then
$\mathit{Concept}(e) = \exists \mathit{Role}(e).\mathsf{e}$
If *e* has $\mathit{minOccurs} = n$ then
$\mathit{Concept}(e) = \exists^{\geq n} \mathit{Role}(e).\mathsf{e}$
If *e* has $\mathit{maxOccurs} = n$ then
$\mathit{Concept}(e) = \exists^{\leq n} \mathit{Role}(e).\mathsf{e}$
If *e* has $\mathit{minOccurs} = n$ and $\mathit{maxOccurs} = m$ then
$\mathit{Concept}(e) = \exists^{\geq n} \mathit{Role}(e).\mathsf{e} \sqcap \exists^{\leq m} \mathit{Role}(e).\mathsf{e}$

In some cases, it may be possible to omit the cardinality constraints in the schema, but rather specify it in the body of the form. As such, when the cardinality constraints of the element is not specified in the schema, it is necessary to look into the corresponding form control in the body of the document. In cases, where the cardinality constraints of the element is not defined neither in the schema nor the body of the form, we have to use value resticition to describe the relationship between the complex element and the corresponding child.[3]

Definition 3 The function *Role* maps elements of an XForms document to roles in $\mathcal{ALCOQI}$. Let *e* be an element of an XForms document, then

$$\mathit{Role}(e) = \mathsf{child\text{-}e}$$

The `nodeset` attribute of a `<bind>` element is used to bind an instance to the binding expression via an XPath. There are two ways to specify XPaths: an XPath expression can be either an absolute or arelative path expression. When encountered binding element via relative path expression, each child element corresponding to the path must be defined.

According to Definitions 1, 2 and 3 the structure of the TBox of an XForms document is created but in a "weak" state. XPath and `<bind>` element are then used to include additional information to strengthen the TBox. An XPath expression contains meaningful information on the elements or the values it is pointing to. This information is used to interpret its related concepts or domains. In addition, XPath and binding elements are also used to verify or enhance the roles and quantifiers established by Definition 2. Most frequently declarations of XPath are found in the tags of binding elements, form controls and at times

[3]Here we would like to point out a conflict between the specifications for XML Schema and XForms. If an element is defined in the schema without any cardinality constraints then the specification for the XML Schema assume the default value $\mathit{minOccurs} = 1$, on the other hand the specifications for XForms state that the defeault value for the attribute required is false(). However *minOccurs* can be defined only in the schema while required only in the other part of an XForms document.

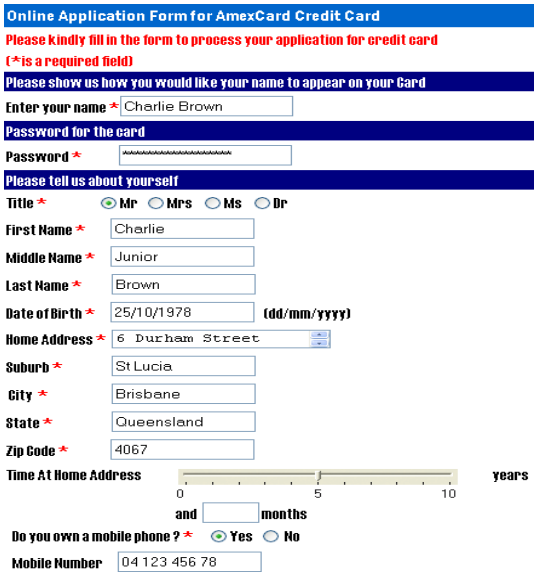

Figure 3: Form displaying additional field mobileNumber when user selected 'Y' as ownMobile's value.

in XML schema. In the `<bind>` element, an XPath expression is used for various binding expression such as model binding expression, UI or action binding expression and computed expression. In this paper, we focus on the model, action and relevant of computed expression. Although it is possible to capture the semantics of the rest of the computed expression, it is not within the scope of this paper.

Definition 4 The functions *XPathE* and *XPathV* take as input an XPath expression x and return, respectively, a set of elements and a set of values according to the following algorithm

$XPathE(x) := \emptyset$
$XPathV(x) := \emptyset$
For each node t in the document compare x with the path of t
If x matches an element e then
  $XPathE(x) := XPathE(x) \cup \{e\}$
If x matches an value v then
  $XPathV(x) := XPathV(x) \cup \{v\}$

The element `<amexcard>` in Figure 1 is a complex concept. It has three child elements located inside the sequence tag. The first two elements, `<fullName>` and `<password>`, are atomic concepts and the last element, `<personalInfo>`, is a complex concept.

According to Definition 1, the partial mapping of the complex concept amexcard is described as:

$$\begin{aligned}\text{amexcard} \sqsubseteq\ & \mathit{RoleConcept}(\mathit{fullName}) \\ & \sqcap \mathit{RoleConcept}(\mathit{password}) \\ & \sqcap \mathit{RoleConcept}(\mathit{personalInfo})\end{aligned}$$

The children of `<amexcard>` do not contain the attribute `relevant`, thus all *RoleConcept* in the above definition turn out to be *Concept* functions.

The elements `<fullName>` and `<password>` have no cardinality constraint specified in the schema or the body of the form, therefore the quantification for the role must be universal. The element `<personalInfo>` has $\mathit{maxOccurs} = 1$, hence number restriction must be used.

According to on Algorithm 2, the mapping of complex concept amexcard is described as follows:

$$\begin{aligned}\text{amexcard} \sqsubseteq\ & \forall \text{child-fullName.fullName} \\ & \sqcap \forall \text{child-password.password} \\ & \sqcap \exists^{\leq 1} \text{child-personalInfo.personalInfo}\end{aligned}$$

The binding statements in Figure 1 specify that the elements `<fullName>` and `<password>` are required fields via the attribute `required` and nodeset. The nodeset points to elements in the schema/document; the attributes are used to indicate the type of relationships between the node corresponding to the Xpath expression and its parent element. The representation of the `<bind>` elements for `<fullName>` and `<password>` gives us the following two assertions

$$\begin{aligned}&\text{amexcard} \sqsubseteq \exists \text{child-fullName.fullName} \\ &\text{amexcard} \sqsubseteq \exists \text{child-password.password}\end{aligned}$$

Consequently the complete definition of amexcard is

$$\begin{aligned}\text{amexcard} \sqsubseteq\ & \forall \text{child-fullName.fullName} \\ & \sqcap \exists \text{child-fullName.fullName} \\ & \sqcap \forall \text{child-password.password} \\ & \sqcap \exists \text{child-password.password} \\ & \sqcap \exists^{=1} \text{child-personalInfo.personalInfo}\end{aligned}$$

Element `<homeAddress>` in Figure 1 has five children of which suburb is declared as an attribute. As explained before, an attribute is treated as an element. The attribute `suburb` is of built-in data type and thus suburb is atomic concept. The `<title>` element does not have a data type specified in the schema. The user therefore has to look into the corresponding field of the element in the body to determine the data type. The `<title>` element defines an atomic concept as its date type is of user-defined but simple type as no elements are specified within it. There are 4 values namely Mr, Mrs, Ms and Dr inside the `<choice>` tag associated with `<title>`. Among these values, only one is selected as the value for `<title>` element (*selectone* form control), therefore resulting in a cardinality constraint of *minOccur is 1*. The mapping of the atomic concept title is described as:

$$\text{title} \sqsubseteq \{\text{Mr}, \text{Mrs}, \text{Ms}, \text{Dr}\}$$

We use the last binding statement in Figure 1 to illustrate the mapping of the attribute `relevant`. First of all the element `<personalInfo>` generates the following expression:

$$\begin{aligned}\text{personalInfo} \sqsubseteq\ & \forall \text{child-name.name} \\ & \sqcap \forall \text{child-homeAddress.homeAddress} \\ & \sqcap \forall \text{child-ownMobile.ownMobile} \\ & \sqcap \forall \text{child-mobileNumber.mobileNumber}\end{aligned}$$

The element `<mobileNumber>` depends on the value of element `<ownMobile>`. The expression we obtain from this is:

$$\begin{aligned}\text{personalInfo} = (&\text{ownMobile:}\{Y\} \sqcap \\ &\exists \text{child-mobileNumber.mobileNumber}) \\ &\sqcup \neg \text{ownMobile:}\{Y\}\end{aligned}$$

Figure 3 shows the form of Figure 2 with the additional field `<ownMobile>` when the user keyed in 'Y' into the value of element `<ownMobile>`. Subsequently the interpretation of the filler ownMobile:$\{Y\}$ will be satisfied by the instance of the concept personalInfo, and at the same time negation of the filler is false. Accordingly the resulting expression can be embedded in the full definition of personalInfo.

$$\begin{aligned}\text{personalInfo} \sqsubseteq\ & \exists \text{child-name.name} \\ & \sqcap \exists \text{child-homeAddress.homeAddress} \\ & \sqcap \exists \text{child-ownMobile.ownMobile} \\ & \sqcap \text{ownMobile:}\{Y\} \\ & \sqcap \exists \text{child-mobileNumber.mobileNumber}\end{aligned}$$

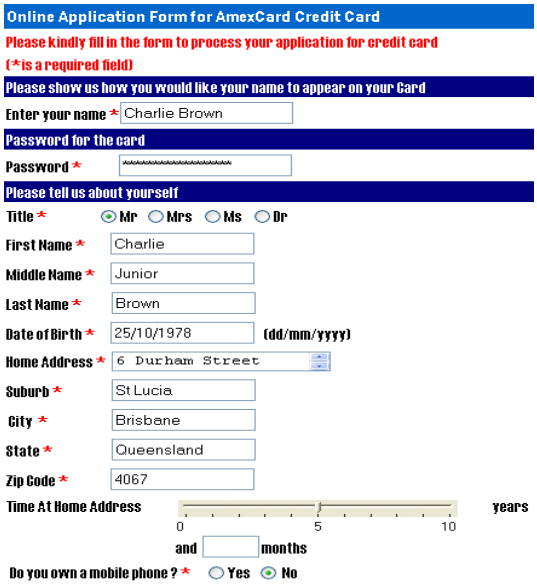

Figure 4: Form displaying without field mobileNumber when user selected 'N' as ownMobile's value.

Figure 4 shows the form of Figure 2 when the user keyed in 'N' instead of 'Y' into the value of the element `<ownMobile>`. Subsequently the interpretation of ownMobile:$\{Y\}$ is not satisfied. This would make the whole conjunction false and the expression which would be returned is:

$$\text{personalInfo} \sqsubseteq \neg \text{ownMobile:}\{Y\}$$

and embedded into the original concept personalInfo as shown below:

$$\begin{aligned}\text{personalInfo} \sqsubseteq\ & \exists \text{child-name.name} \\ & \sqcap \exists \text{child-homeAddress.homeAddress} \\ & \sqcap \exists \text{child-ownMobile.ownMobile} \\ & \sqcap \neg \text{ownMobile:}\{Y\}\end{aligned}$$

4.2 Structuring the ABox

The body of the form contains form controls that are used to structure the ABox in a knowledge base and for specifying the user interface of the form. Only form controls with attribute `ref` and that receive input from users are used in structuring the ABox. Form controls with attribute `ref` are `<input>`, `<secret>`, `<textarea>`, `<select1>`, `<select>` and `<range>`. Attribute `ref` references an element in the instance document, which also belongs to a concept in the TBox. As such, the value of the form control denotes the value of that concept (concept assertions).

The first three form controls, namely `<input>`, `<secret>` and `<textarea>`, are simple and straightforward when used for structuring the ABox but the remaining controls, namely `<select1>`, `<select>` and `<range>`, need additional steps. Both the selection form control (`<select1>` and `<select>`) provides a list of items, which are defined by using `<label>` and `<value>` tags. The former tag gives the description of an item while the latter is the value associated with the item. It is noted that in most cases `<value>` tag do not carry significant information about its corresponding `<label>` tag and therefore, it is necessary to take into consideration both tags when determined the domain of that concept.

In `<range>` form control, the referencing of the domain for a concept is determined by its attributes, namely `start`, `end` and `step`. The `start` attribute provides the lexical starting bound for the range and denotes the first value, while the `end` attribute gives the ending bound for the range and denotes as the last value of the domain. The `step` attribute is applied to increment or decrement the value, and is used to determine the values inside the range.

As discussed, the role represents the relationship between the elements in a complex concept. By using role assertions, one can state that c is a filler of the role R for b, where b is the ID of the root element of a complex concept and that c is the value (interpretation) of a concept. This is illustrated as:

$$R(b,c).$$

For instance,

$$\text{child-name}(\textit{amexcard123}, \textit{Charlie}),$$

where *Charlie* is the value of element `<firstName>`, implies that *amexcard123* has a first name's field, whose value is *Charlie*. XML Schema uses attribute names to represent elements but in some instances, there may be more than one element with the same name. For instance, an element named person with cardinality constraint of *maxOccurs=2*. As such, a unique ID is assigned to each element. This ID aims to identify each unique element and is derived from the attribute name of the element. In the case of elements with duplicate name, the ID will still inherit the name of the element but will append a numbering order at the end of the ID. In this case, the first element's ID is person1 while the latter element's ID is person2.

5 Perform Reasoning Tasks

In the previous section, we have defined and illustrated the methodology of structuring the knowledge base of an XForms document. In this section, we will explain the reasoning tasks one can perform on XForms documents. Foremost, the reasoning on validity and logical design of form as specified in Section 2.2 can immediately be reduced to the fundamental problem of checking the satisfiability of the knowledge base's domain. This verification of satisfiability is performed via using the reasoning tasks available in $\mathcal{ALCOQI}$.

Consistency of forms

Once the schema and the binding elements of the form have been mapped into the knowledge base, each mapped binding element is checked whether the mapped binding element is embeddable into its associated mapped schema's element as shown below:

$$\textit{Mapped}(\textit{binding}) \sqsubseteq \textit{Mapped}(\textit{schema})$$

The data of the field is correctly interpreted by its corresponding field's data in a form if the interpretation of $\textit{Mapped}(\textit{binding})$ is a subset of the interpretation of $\textit{Mapped}(\textit{schema})$.

Let us consider for example the following (partial) Knowedlge Base.

$$\begin{aligned}\text{personalInfo} \sqsubseteq\ & \exists \text{child-name.name} \\ & \sqcap \exists \text{child-homeAddress.homeAddress} \\ & \sqcap \exists \text{child-ownMobile.ownMobile} \\ & \sqcap ((\text{ownMobile:}\{Y\} \\ & \ \sqcap \exists \text{child-mobileNumber.mobileNumber}) \\ & \ \sqcup \neg \text{ownMobile:}\{Y\})\end{aligned}$$

This Knowledge Base has an element ownMobile contained in the complex concept personalInfo, which the user has chosen 'Yes' as the value for ownMobile. Assuming in the form, there is a particular field displaying

the value of ownMobile as 'No'. The interpretation of this particular field will be reflected as:

$$\mathsf{ownMobile}^{\mathcal{F}} = \{No\}$$

Based on *consistency of forms* definition, this particular field is verify whether it could be embedded into the mapped schema as:[4]

$$\mathsf{personalInfo}^{\mathcal{F}} \sqsubseteq \mathsf{personalInfo}^{\mathcal{S}}$$

The representation of the field could not embedded into the mapped schema, as the interpretation of the mapped field $\mathsf{OwnMobile}^{\mathcal{F}}$ is 'No' which is not equivalent to the interpretation 'Yes' of OwnMobile of the mapped schema $\mathsf{personalInfo}^{\mathcal{S}}$. Therefore the data of this field is not consistent with its corresponding element's data, making the form invalid or inconsistent for the user.

Embedding of forms

Given two forms *A* and *B*, form *B* can be embedded in form *A* if

$$Mapped(formA) \sqsubseteq Mapped(formB)$$

One form can be embedded into another if the interpretation of $Mapped(formB)$ is a subset of the interpretation of $Mapped(formA)$.

Assuming there is an enrollment task, which requires the user to fill in two forms, A (application form for joining a course) and B (application form for newsletter that is relevant to that course). Form A requirea applicant personal details such as name, residential and company address while Form B requirea applicant name only. Both form A and B have some identical fields namely `<firstName>`, `<middleName>` and `<lastName>` as shown below:

Partial Knowledge Base of Form A

$$\begin{aligned}
\mathsf{name} \sqsubseteq\ & \exists \mathsf{child\text{-}title.title} \\
& \sqcap \exists \mathsf{child\text{-}firstName.firstName} \\
& \sqcap \exists \mathsf{child\text{-}middleName.middleName} \\
& \sqcap \exists \mathsf{child\text{-}lastName.lastName} \\
\mathsf{title} \sqsubseteq\ & \{\mathrm{Mr}, \mathrm{Mrs}, \mathrm{Ms}, \mathrm{Dr}\} \\
\mathsf{firstName} \sqsubseteq\ & \mathsf{string} \\
\mathsf{middleName} \sqsubseteq\ & \mathsf{string} \\
\mathsf{lastName} \sqsubseteq\ & \mathsf{string}
\end{aligned}$$

Knowledge Base of Form B

$$\begin{aligned}
\mathsf{name} \sqsubseteq\ & \exists \mathsf{child\text{-}firstName.firstName} \\
& \sqcap \exists \mathsf{child\text{-}middleName.middleName} \\
& \sqcap \exists \mathsf{child\text{-}lastName.lastName} \\
\mathsf{firstName} \sqsubseteq\ & \mathsf{string} \\
\mathsf{middleName} \sqsubseteq\ & \mathsf{string} \\
\mathsf{lastName} \sqsubseteq\ & \mathsf{string}
\end{aligned}$$

Based on *embedding of forms* definition, we have to check whether form *B* can be embedded in form *A*, namely

$$\mathsf{name}^{\mathcal{A}} \sqsubseteq \mathsf{name}^{\mathcal{B}}$$

where $\mathsf{name}^{\mathcal{B}}$ is form *B* and $\mathsf{name}^{\mathcal{A}}$ is form *A*. The complex concept name in form *A* contains elements firstName, middleName and lastName. The interpretation of these elements is identical to the interpretation of form B's elements because both identical element's value is equivalent. The identical fields in form B can be embedded into the mapped schema of Form A. Form B is therefore embeddable into form A thus reducing the number of forms for the user to fill.

Form fillability:

Once the schema and the binding elements of the form have been mapped into the knowledge base, the interpretation of domain of the embedded binding element and schema's element is checked for emptiness as shown below:

$$Mapped(binding) \sqcap Mapped(schema) \sqsubseteq \bot$$

The data of the field is relevant to the semantic of the form if the embedded binding element and schema's element has a non-empty interpretation.

In the Knowledge Base

$$\begin{aligned}
\mathsf{personalInfo} \sqsubseteq\ & \exists \mathsf{child\text{-}name.name} \\
& \sqcap \exists \mathsf{child\text{-}dob.dob} \\
& \sqcap \exists \mathsf{child\text{-}homeAddress.homeAddress} \\
& \sqcap \forall \mathsf{child\text{-}timeHAddr.timeHAddr} \\
& \sqcap \exists \mathsf{child\text{-}ownMobile.ownMobile} \\
& \sqcap \neg \mathsf{ownMobile}{:}\{Y\}
\end{aligned}$$

there is a element ownMobile contained in complex concept personalInfo, where the user chosen 'No' as the value of ownMobile. Although the user choosen 'No' for the value of ownMobile, in the form there is another field corresponding to mobileNumber, where the user has to input his mobile phone number. Since the user does not own a mobile phone, the field would be empty, namelly:

$$\mathsf{mobileNumber}^{\mathcal{F}} = \emptyset$$

According to the definition of *form fillability*, the interpretation of domain of the field mobileNumber and the concept personalInfo is checked for emptiness as:

$$\mathsf{personalInfo}^{\mathcal{F}} \sqcap \mathsf{personalInfo}^{\mathcal{S}} \sqsubseteq \bot$$

$\mathsf{personalInfo}^{\mathcal{S}}$ does not contain the element mobileNumber and therefore the domain of the $\mathsf{personalInfo}^{\mathcal{S}}$ would not contain any interpretation of mobileNumber. This implies that the interpretation of $\mathsf{personalInfo}^{\mathcal{S}}$ is empty. The conjunction of $\mathsf{mobileNumber}^{\mathcal{F}}$ with $\mathsf{personalInfo}^{\mathcal{S}}$ would resulted in a empty (inconsistent) interpretation. This causes the field corresponding to mobileNumber to be not relevant to the semantics of the form, in turn making the form not fillable or invalid to the user.

Integration of forms with ontologies:

Once the schema and the binding elements of the form have been mapped into the knowledge base, the embedded binding elements and schema's elements are verified against a predefined ontology, provided eventually by an ontology server. Then the interpretation of domain of the embedded binding element and schema's element are checked for emptiness and embedding; that is:

$$Mapped(ontology) \sqcap Mapped(form) \sqsubseteq \bot$$

and

$$Mapped(ontology) \sqsubseteq Mapped(form)$$

[4] $^{\mathcal{F}}$ is used to refer the map of the fields in the form while $^{\mathcal{S}}$ denotes the map of the schema of the form.

The form contains the essential and relevant fields if the embedded binding element and schema's element is admitted by the knowledge base and has a nonempty interpretation. The concept of integration of forms is the combination of form fillability and consistency of forms. Due to the page constraint, the reader is advised to imagine the examples used by form fillability and consistency to understand the concept of integration of forms.

6 Conclusions

The main contribution of the paper is the development of a framework to reason about XForms document based on $\mathcal{ALCOQI}$. We have demonstrated the process of capturing the semantics of the XForms documents and mapping it to knowledge bases in Description Logic. Thereafter reasoning tasks are applied to the knowledge base of the document to verify the validity and logical design of the forms. In addition, we argued that the $\mathcal{ALCOQI}$ is expressively enough for this specific task.

There are several research directions that are worth pursuing. In this paper, we relied on $\mathcal{ALCOQI}$ to represent and reason about forms, other logics such as First Order Logic (FOL) and Propositional modal logics are not taken into consideration. As such, the first research area is using other logics to represent and reason about forms. Comparsion can be made in terms of expressiveness and complexity to determine which is the most appropriate logic to represent and reason about form.The second research area is further aspects of the XPath and binding element could be captured in order to present other properties of the XForms document such as calculate and constraint property. This will challenge the expressive power of $\mathcal{ALCOQI}$ as whether the logic is capable of representing other properties of XForms documents.

The third research area is exploring the generation of web forms with respect to the integration of ontologies [11]. As mentioned in the introduction, schemas can be viewed as ontologies therefore it is possible to generated forms based on the ontologies [7, 5, 8]. The main issue is to study whether the generated forms are consistent with respect to the ontologies, indeed the a form may interoperate with several ontologies thus we have to study how to integrate ontologies to produce a global ontology for the form.

Finally as discussed in the introduction, the proposed reasoning framework could be implemented into the existing DL system such as FaCT system to reason about XForms document with respect to the requirements of the users and applications. Depending on the reuirement of the system using the framework the validation of forms can be carried out either online or offline.

References

[1] Novell xforms strategy. Technical report, Novell XForms Technology Preview, 2003.

[2] Franz Baader, Diego Calvanese, Deborah McGuinnes, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logics Handbook*. Cambridge University Press, Cambridge, 2003.

[3] Franz Baader and Werner Nutt. Basic description logics. In Baader et al. [2], chapter 2, pages 43–95.

[4] Alex Borgida and Ronald J. Brachman. Conceptual modelling with description logics. In Baader et al. [2], chapter 10, pages 349–372.

[5] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Information integration: Conceptual modelling and reasoning ssupport. In *CoopIS'98*, pages 280–291, 1998.

[6] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Representing and reasoning on XML documents: A description logic approach. *Journal of Logic and Computation*, 9(3):295–318, 1999.

[7] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Description logic framework for information integration. In *6th International Conference on Principles of Knowledge Representation and Reasoning (KR'98*, pages 2–13, 1998.

[8] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Daniele Nardi, and Riccardo Rosati. Data integration in data warehousing. *International Journal of Cooperative Information Systems*, 10(3):237–271, 2001.

[9] Diego Calvanese, Maurizio Lenzerini, and Daniele Nardi. Description logic for conceptual data modeling. In Jan Chomicki and Gunter Saake, editors, *Logics for Databases and Information Systems*, chapter 8, pages 229–263. Kluwer, Norwell, MA, 1998.

[10] W3C Consortium. *XForms*. W3C Consortium, August 2003.

[11] Marlon Dumas, Lachlan Aldred, Mitra Heravizadeh, and Arthur H.M. ter Hofstede. Ontology markup for web forms generation. In *Workshop on Real World RDF and Semantic Web Applications*, May 2002.

[12] Ian Horrocks. Using an expressive description logic: FaCT of fiction. In *Proceeding 6th International Conference on Principles of Knowledge Representation (KR'98*, pages 636–647. Morgan Kaufmann, 1998.

[13] Joel Rivera and Len Taing. Get ready for xforms. Technical report, IBM, September 2002.

[14] Sebastin Schnitzenbaumer. *XForms: The next generation of Web technology*. Software AG - The XML Company.