Spring 6-20-2012

Exploiting XML Technologies in Medical Information Systems

Christian Forster
University of Münster, Germany, christian.forster@ercis.uni-muenster.de

Gottfried Vossen
University of Münster, Germany, vossen@uni-muenster.de

25th Bled eConference
eDependability:
Reliable and Trustworthy eStructures, eProcesses, eOperations and eServices for the Future
June 17, 2012 – June 20, 2012; Bled, Slovenia

Exploiting XML Technologies in Medical Information Systems

Christian Forster
University of Münster, Germany
christian.forster@ercis.uni-muenster.de
Gottfried Vossen
University of Münster, Germany
vossen@uni-muenster.de

Abstract
Integration of clinical research data and routine care data, in order to streamline the process of conducting clinical studies, has been a problem for quite a while now. The Single Source project at the University of Münster aims at contributing to this area. The approach is based on a vast usage of XML technology together with a novel integration architecture. The emphasis in this paper is on the former: The seamless usage of XML technology throughout the entire application is presented, and mismatches of programming paradigms are averted by exploiting the features of XML, XQuery and XForms. In particular, this is demonstrated by the example of a component used for handling forms, by how it is built and used in the entire scenario.

Keywords: XML, XQuery, XForms, software engineering

1 Introduction

Source integration in medical information systems has been a problem for quite a while now, since typically clinical research data as well as routine care data are collected, stored, and maintained independently of each other [Dugas et al., 2009]. An immediate consequence is that utilizing data from one source in conjunction with data from the other, for example in the context of a given clinical study for which this would be highly beneficial, is difficult, if not impossible. The Single Source project aims at changing this situation fundamentally; its basic approach is based on a vast usage of XML technology, together with a novel integration architecture. The emphasis in this paper is on the former; it is demonstrated how the form handler is built and used in the entire scenario.

The Single Source Project is a research project conducted at the authors' institute in Germany and the software design presented here has evolved from this. The project

aims to provide a flexible platform that is independent of a particular hospital information system and through which care data available for patients in a hospital can be integrated with research data. In our approach, the actual integration is based on electronic forms. These forms are filled with default values originating medical knowledge, completed with care data, and then extended with research data, yielding a comprehensive data set for evaluation. The software design presented here reflects our experience with the development of the form handler. The form handler is part of the x4T system architecture [Dziuballe et al., 2011], designed to store form definitions and their associated form data, and to perform operations on it. This work presents a software design used in the domain of clinical trials and shows how this can benefit from a homogeneous XML software stack composed of W3C standards.

Object orientation is often seen as the state-of-the-art in present-day technology, which is why we initially considered following an object oriented approach. However, we abandoned this for the following reason: The data collected in clinical trials is given by a set of data capture forms that have to be filled in for every patient participating in a given study. These forms consist of a hierarchy of elements representing a study at the highest level and a single item at the lowest. Breaking down these elements into objects would have resulted in a vast set of objects and relationships between them to represent a study, and that would have been difficult and tedious to handle. Therefore, we consider the domain elements of clinical studies not to be candidates that are appropriately modeled the object oriented way. Although the paradigm definitely has its benefits in many cases and application areas [Bhattacharya and Neamtiu, 2011], it is not necessarily the best choice in terms of productivity [Myrtveit and Stensrud, 2008]. What makes it even worse is the fact that there is a paradigm shift when using relational DBMS as data layer and HTML forms based on key-value pairs in the presentation layer, in combination with an object oriented business layers which is an often proposed technical stack. Many more or less heavy-weight frameworks and mappers address these paradigm transformation issues, thereby complicating the code by adding layers and solving problems related to paradigm mismatch instead of the given domain tasks at hand.

Clinical trials are very data centric, and the standard exchange format for forms is CDISC ODM (the *Operational Data Model*, a standard for the transfer of case report form data developed by the *Clinical Data Interchange Standards Consortium*) which is XML-based and comprises both form definition and data. Forms for trials must be generated based on those definitions, and besides CRUD operations (short for “create, read, update, delete”) there is need for business functions in order to pre-populate forms with values from routine care systems, and for administrative functions like access protection and user management. These considerations made us design and implement our XML-based approach using a mostly functional programming paradigm, in order to avoid the aforementioned pitfalls.

The remainder of this paper is organized as follows: Section 2 gives an overview of related work. Section 3 highlights key components of the XML-based architecture and describes their generic design. In Section 4 we describe scenario-specific solutions of the form handler, and Section 5 concludes with an outlook on future work.

2 Related Work

XML and the various standards and languages based on it have been used for quite a while now in data-intensive applications; we here review work only that can be considered relevant to the context of medical information systems and to our work. Approaches for implementing business logic in XQuery have been undertaken in [Kaufmann and Kossmann, 2009], where the development of an online publication repository is described. The authors state that XQuery was well suited, but maturity of application servers was the biggest concern and that further experience with other applications was needed. We took this as starting position to build on and extended the approach by the usage of XForms and adapted it to our needs.

The label XRX refers to the idea of using XForms, REST and XQuery. The XRX concept is described and discussed on the Web [McCreary, 2007, Cagle, 2008, Wikibooks, 2010]. Though no sharp classification seems to be generally accepted, offering REST access— an architectural style relying on HTTP commands containing all state related information— to data seems to be crucial. As our application does not offer a pure REST interface, we do not consider it to be classified as XRX, but some parts of the concepts overlap.

A recently published approach is using Java and the Spring Framework to embrace XML related technologies offering RESTful services in the field of healthcare [Davis and Maguire, 2011]. Although their domain is closely related to ours, we have purposely avoided the usage of object oriented Java and a framework like Spring. Finally, the industry consortium IHE offers a set of specifications in order to make healthcare systems interoperable, among them the Retrieve Form for Data Capture [IHE International, 2009]. This specification proposes the use of XForms as form definition and exchange standard.

For the particular task of form design and handling, various commercial products are available, including Adobe LiveCycle and Microsoft Infopath. However, we have deliberately excluded solutions that are built on a proprietary software stack from our considerations.

3 The Architectural Approach

This section highlights key components of the XML-based architecture and describes their generic design.

3.1 The Components: XML Family as Technical Base

XML and related technologies found the basis of the XML-based architecture; this section points out the most relevant technologies and their characteristics.

3.1.1 XML

XML [Bray et al., 2008] is a framework for markup languages often called semi-structured, meaning the data is embedded in its metadata. The term semi-structured is to some extent misleading [Sperberg-McQueen, 2005], as the data is completely structured, but in a flexible rather than a static way. In our case, the structure representation in XML is an important feature. XML documents are structured as trees and thus suit hierarchically organized data well. XML schemata are used for defining the specific structure of XML documents. Though not necessarily required, schemata

ensure that the XML document is structured according to defined rules. The schema used for documentation of clinical trials, ODM [CDISC, 2010a], is both a format to design the study and to store its data. The support of namespaces allows for vendor-specific extensions in a separate namespace; thus we can both gain compatibility benefits from supporting a standardized schema and implement new features.

3.1.2 XQuery

XQuery is a Turing complete functional programming language [Kepser, 2004] that is intended to work on XML documents and that has been specified by the W3C [Boag et al., 2010]. Having its origin in querying XML, update expressions were not standardized by W3C for long, but since 2011 the XQuery Update Facility 1.0 [Robie et al., 2011] is available and has started to replace various vendor specific mechanisms. At least since then, XQuery has evolved beyond being a simple data query language. To us, it has shown to be a practically usable language supported by a sufficient number of libraries and nowadays even stable application servers. In 2009 the absence of those impeded productive usage of XQuery [Kaufmann and Kossmann, 2009].

XQuery supports the functional paradigm to a vast extend, though it is not purely functional. Basically, a program written in XQuery is composed of functions. The output of a function is solely depending on its input, hence there is no internal state of the program and there are no side effects of a function. Along with these features come several benefits: Testing of decomposed functions is easy, since no initialization is required and a function can be tested by a set of input parameters and the expected outcome. As functions do not rely on an internal program status, parallel execution of functions is possible without additional synchronization code.

The implementation we have chosen is not purely functional, since it supports reading and writing to external media which reflects a kind of program state. Thus—in contradiction to the pure functional paradigm—some functions have empty return values because their purpose is realized by side effects, e.g., writing data on disk. It is up to the programmer to stick to the functional implementation whenever it is useful and take care of side effects when necessary.

The basic expression in XQuery is the FLWOR expression, an acronym for the keywords `for`, `let`, `where`, `order by` and `return`, used to specify the nodes that operations are based on (`for`), to define variables within that expression (`let`), to filter elements by conditions (`where`), to sort them by criteria (`order by`) and to build the result (`return`).

A subset of the XQuery specification that deals with paths in XML documents is XPath [Berglund et al., 2010]. It provides means to address nodes and to navigate through a given XML structure.

3.1.3 XSLT

Besides XQuery, there is XSLT [Kay, 2007], an earlier proposal for processing XML documents. It has similar features as XQUERY and is also Turing complete [Kepser, 2004], but it follows a different design approach. The original focus was on a transformation of XML documents rather than on querying databases. XSLT uses XML syntax and is intended as a style sheet language based on templates. It has XPath support in common with XQuery.

3.1.4 XForms

Pure XHTML forms do not use a specific approach to data modeling, but submit form data as a set of key-value pairs, which is different from the data model present in most business and data layers. In our case, the model in both business and data layer is XML-based; it is therefore preferable to use the same technique in the presentation layer. The W3C standard XForms [Boyer, 2009] provides a way to embed forms into a host language such as XHMTL. XForms provides several advantages over standard XHTML forms that even the upcoming version 5 does not have. The most important of these is the separation between model, view, and controlling structures.

There are server-side XForms implementations which generate XHTML and JavaScript from XForms. Ideally, this transformation is done automated, and the developer is not concerned with any XHTML or JavaScript forms code. The XForms implementation takes care of synchronization between client and server state, and provides both client (for quick response and comfort) and server (for security because it cannot be bypassed) side validation based on the model definition. Thus, clients do not need any additional XForms interpreter but just a common Web browser, a fact that is relevant to the clinical scenario, where additional software deployment to clients is regulated and arduous.

3.2 The Composition: A Homogeneous Structure

The XML-based architecture proposed here consists of distinct layers for storage, business logic, and presentation. Although the single language approach would allow for a direct access of the data layer from the presentation layer, the multitier architecture is kept for an easy integration of additional presentation techniques and a better reusability. Figure 1 illustrates the various layers and their associated technologies: XQuery is used for the data and business layers, while the presentation layer is implemented in XHTML and XForms.

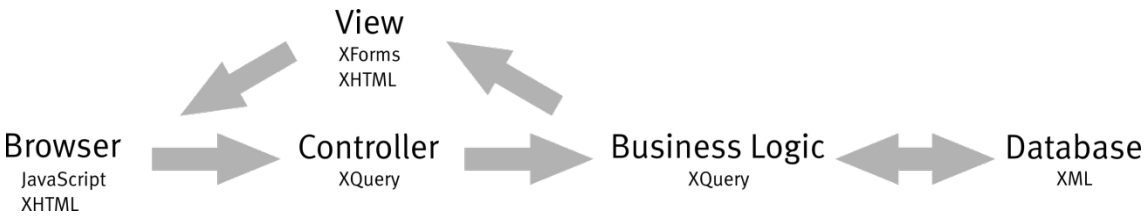

Figure 1: Layers of the XML-based architecture and their techniques.

The business logic consists of two parts. Functions are separated into reusable modules. For every function there is an HTTP-specific fixture called by the controller to transform HTTP parameters into XQuery variables and to call the functions from the modules.

The homogenous XML technology stack allows for usage of a single data model for the storage, business, and presentation layers. Additional patterns for bridging the paradigm mismatch, such as object-relational mappings or patterns of abstraction like DAO [Sun Microsystems, 2001] and DTO [Fowler, 2003], are no longer necessary in this approach and hence do not add complexity. On the storage layer, a native XML database supports document oriented storage and XQuery for data access without a separate layer of shredding and publishing [Tatarinov et al., 2002]. The business logic can directly

operate on the data. No additional layer or transformation is needed, because XQuery is used as both a business logic programming language and a data query language.

The integration of XForms allows using the data schema in the presentation layer. Again, no transformation of the model is needed, and the conversion to XHTML and JavaScript is done by the XForms engine on the server. The XForms capability of supporting a distinct model in the presentation brings the benefit that schema constraints like data types can automatically be bound to the corresponding form elements. Parts of the web application that do not contain forms consist of XHTML templates, where dynamic elements are realized with embedded XQuery expressions.

4 Our case: x4T Form Handler

In this section we describe scenario-specific solutions of the form handler outlined in Section 3.

4.1 Scenario Overview

Our form handler is embedded into a clinical information system landscape that consists of a hospital information system (HIS) and one of several clinical study information systems. Our particular hospital information system in which most of the medical data is kept is Agfa Orbis and an Oracle relational database. Since Orbis does not offer a sufficient interface to access patient data on the business layer, we had to perform integration at the data layer and set up an additional mediator referred to as *HIS handler* to make data accessible. All HIS specific implementations are encapsulated in the mediator. It offers Web services to gather data and authentication information.

As medical staff should be able to access the trial documentation from Orbis, which is their daily working environment, access to x4T forms is given by dynamically created Web links. One-time access tokens are created by the HIS handler and written to the database, from which they are fetched by Orbis and encoded within those links. The x4T form handler, in particular the controller (cf. Figure 1), extracts the token and checks its validity using the authentication Web Service. Doing so, no manual user authentication is needed and access grants in HIS and x4T are consistent. Updated patient data is integrated into the form handler from the HIS before every access to a patient's study documentation, in order to ensure that staff has a consistent view and can see all recent entries from the routine documentation.

While filling the form, the researcher decides which of the pre-populated values are correct or whether a completely different value, entered manually, is needed. The integration is based on semantic annotations describing medical phenomena that are present within the study form definition and can be resolved into SQL queries by the HIS handler. Note that a fully automated integration is mostly not possible, due to often complex requirements beyond the pure item value and its origin, so that expert knowledge is needed to decide on the correctness of an entry. The form handler allows an export of study data into common study information systems (CDMS) via the ODM [CDISC, 2010a] format. Figure 2 shows the components and their interfaces.

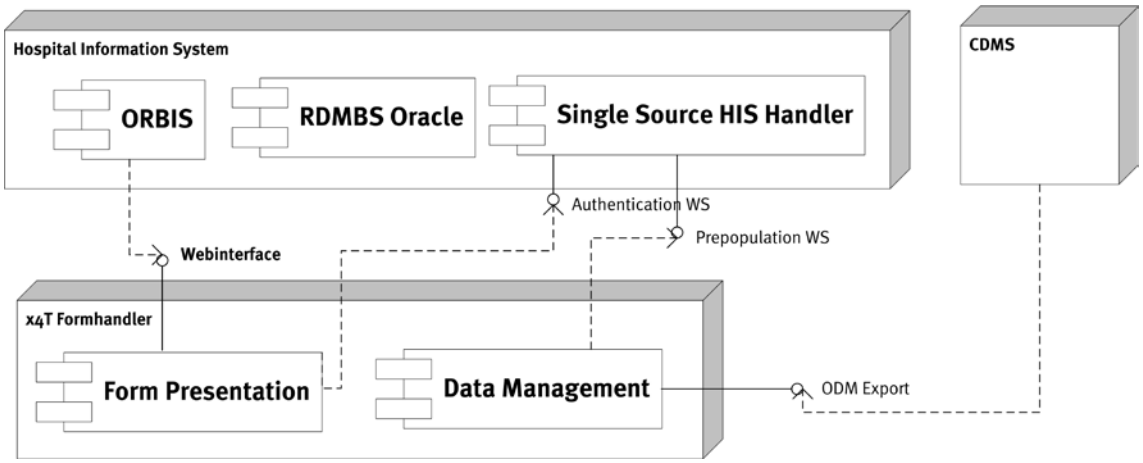

Figure 2: Components of the Single Source scenario.

4.2 Towards Practical Usage of x4T

The standard that is supported by a large number of vendors to exchange clinical trial data and metadata is CDISC ODM. It is freely available, defined by an XML schema, well documented, and open to vendor-specific extensions. Thus, it supports the integration approach of the Single Source project and is used in an initial practical exploitation of x4T to store all study-related data. The ODM schema consists of four complex elements at the first level: Study, AdminData, ReferenceData, and ClinicalData. Study and ClinicalData are most important, as they hold the metadata and data. They are structurally similar: Study holds the definitions of elements comprising references to subordinated elements, which is indicated by suffixes *Ref* and *Def*. ClinicalData holds the actual data, and thus element names are suffixed with *Data*. Listing 1 shows their structure by way of a simplified example. The schema actually stores more information, such as descriptions, measurement units, value ranges etc., using child elements and attributes. The study documentation consists of hierarchically arranged elements, which is reflected in the schema as follows: The Study element contains a MetaDataVersion element that is used to distinguish several versions of the same study. Within the MetaDataVersion element, there is one Protocol element that can have one or more references to StudyEvents. StudyEvents are a group of Forms that belong to a clinical concept. Each Form reflects a traditional paper form and consists of ItemGroups. ItemGroups collect Items of a similar type. Items correspond to a single question. Every element has a unique object identifier (OID) by which it is referenced. Element definitions of all hierarchies starting with the StudyEvent can be reused by referencing several times. The ClinicalData element holds the collected study data for a single MetaDataVersion. For every patient included, there is a SubjectData element that holds a nested set of Data elements, consistent with the metadata defined above. Thus, ItemData is identified by the path of ItemGroupData, FormData and StudyEventData that it is nested in. To give an overview, the data model of patient data in ODM is as depicted in Figure 3.

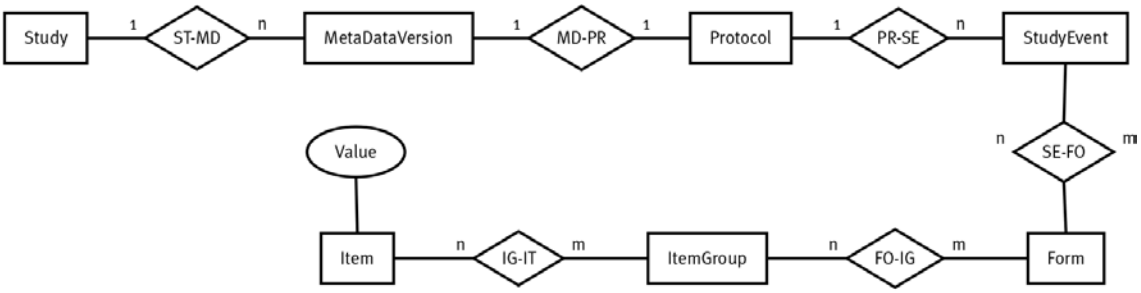

Figure 3: Basic parts of the data model representing a study in ODM.

```
<ODM>
  <Study OID="ST.1" >
    <MetaDataVersion OID="MD.1" >
      <Protocol>
        <StudyEventRef StudyEventOID="SE.1" /><!-- refers to
StudyEvent, multiple allowed -->
      <Protocol>
      <StudyEventDef OID="SE.1" >
        <FormRef FormOID="FO.1" /><!-- refers to Form, multiple
allowed -->
      </StudyEventDef>
      <FormDef  OID="FO.1" >
        <ItemGroupRef ItemGroupOID="IG.1" /><!-- refers to Item,
multiple allowed -->
      </FormDef>
      <ItemGroupDef OID="IG.1" >
        <ItemRef ItemOID="IT.1" />
      </ItemGroupDef>
      <ItemDef OID="IT.1" />
  </MetaDataVersion>
  </Study>
  <ClinicalData StudyOID="ST.1" MetaDataVersionOID="MD.1" >
    <SubjectData SubjectKey="Subject1" ><!-- one for each subject -->
      <StudyEventData StudyEventOID="SE.1" ><!-- refers to defined
StudyEvent -->
        <FormData FormOID="FO.1" ><!-- refers to defined Form -->
          <ItemGroupData ItemGroupOID="IG.1" ><!-- refers to defined
ItemGroup -->
            <ItemData ItemOID="IT.1" Value="Item Value" /><!--
refers to defined Item -->
          </ItemGroupData>
        </FormData>
      </StudyEventData>
    </SubjectData>
  </ClinicalData>
</ODM>
```

Listing 1: Simplified ODM showing Study Definition and Clinical Data Section.

Each study is contained in one ODM file; setting up a study in the form handler is done by importing the corresponding ODM file. To store pre-population data, we have extended the original ODM schema. In compliance with the CDISC specification for

vendor extensions, all extensions happen in a separate namespace and can be removed on export.

```
declare function x4tPat:addPatient($subject-key, $study-oid,
$metadataversion-oid){
  (: some checks on parameters omitted :)
  (: check if patient already exists in study :)
  if (collection('/db/x4t/')/odms/ODM/ClinicalData[@StudyOID = $study-
oid]/SubjectData[@SubjectKey = $subject-key]) then <error>Patient no.
{$subject-key} already existent</error>
  else
  (: create SubjectData :)
  (
  local:build-subjectData(collection('/db/x4t/')/odms/ODM[Study/@OID =
$study-oid], $study-oid, $subject-key, $metadataversion-oid),
  <success>Forms for patient no. {$subject-key} generated</success>
  )
};

(: create the SubjectData XML structure for one subject :)
declare function local:build-subjectData($odm, $study-oid, $subject-
key, $metadataversion-oid){
  (: omitted: create empty ClinicalData node if not already present :)
  (: use XSLT to generate a SubjectData :)
  let $insertion := transform:transform($odm/Study,
xs:anyURI("../subjectDataGenerator.xsl"), <parameters><param
name="SubjectKey" value="{$subject-key}"/><param
name="MetaDataVersionOID" value="{$metadataversion-
oid}"/></parameters>)
  return
    update insert $insertion into $odm/ClinicalData[@StudyOID=$study-
oid and @MetaDataVersionOID=$metadataversion-oid ]
};
```

Listing 2: Adding a subject to ODM as an example for Business Logic realized in XQuery.

Listing 2 shows subject adding as an example for business logic in XQuery. The code consists of two functions: The first checks if adding this subject is possible by querying the database for a subject with the same subject-key in the given Study, and returns an XML fragment indicating the error in that case. Otherwise the second function, that builds the SubjectData, is called. This function makes use of an XSLT style sheet that generates the XML nodes for the subject's data according to the definition given in the Study element. Values for SubjectKey and MetaDataVersion are passed as parameters to the style sheet. Afterwards, the generated SubjectData node is written to the ClinicalData element. Notice the tight data layer integration in XQuery, accessing the database via the `collection` function, and the lean code of applying a transformation and writing back to the database. Unlike the traditional stack, there is no conversion of relational schema to objects, neither implicit by a framework nor explicit by coding.

```
<xsl:template match="Study">
  <html>
    <head>
      <title>
        <xsl:value-of select="$txtStudyID"/><xsl:text>
</xsl:text><xsl:value-of select="$study-oid" />;
```

```
          <xsl:value-of select="$txtPatientID"/><xsl:text>
</xsl:text><xsl:value-of select="$patient-oid" />
       </title>
       <xf:model>
         <xf:instance id="data_instance"
xmlns="http://www.cdisc.org/ns/odm/v1.3"
xmlns:odm="http://www.cdisc.org/ns/odm/v1.3"
resource="/exist/x4t/patient/getClinicalData.xql?{$URLparameters}"
method="get"/>
       </xf:model>
     </head>
     <body>
       <xsl:apply-templates
select="MetaDataVersion[@OID=$metadataversion-oid]">
     </body>
   </html>
</xsl:template>
...
<xsl:template name="itemInput">
   <xsl:param name="storagePath"/>
     <xsl:variable name="destination"><xsl:value-of
select="$storagePath"/>/odm:ItemData[@ItemOID='<xsl:value-of
select="@OID"/>']/@Value</xsl:variable>
   <xsl:if test="@DataType='text' or @DataType='string'">
     <xf:textarea ref="{$destination}" />
   </xsl:if>
</xsl:template>
```

Listing 3: XSLT style sheet to generate the XForms form presentation.

The generation of forms for data capture by researchers is a main aspect of the form handler. The XML basis allows the usage of a single model through all layers and concerning the presentation layer, the ODM patient-specific `SubjectData` element is used as the XForms' model. The user interface consists of XForms code embedded in XHTML that is generated based on the ODM `Study` element defining the study. The generation is done via XSLT. For every element within `Study`, there is a matching XSLT template to construct the corresponding view elements, e.g., `Study` is used to generate the page header, containing information about the study. `StudyEvent`, `Form` and `ItemGroup` are used to generate further grouping, where an `ItemGroup` results in an XHTML table containing items row-wise. Items are converted to XForms elements that support input based on their data type, e. g., textual input fields for strings and date picker for dates. Listing 3 shows parts of the XSLT template, which in total consists of nearly 900 lines, to illustrate the way it works. The `xsl` namespace contains XSLT elements, the `xf` namespace is for XForms elements and the default namespace is XHTML. The `xf:model` element contains the XForms model, in this case the instance is loaded from an URL. The `apply-templates` element is the entry point of the ODM traversal that results in calling the `template` element named `itemInput`. The given example shows the `xf:textarea` element that is rendered if the data type is plain text.

The link to the corresponding ItemData element is generated along the template calling path and handed over in the variable `storagePath`: As the template traverses the hierarchy of the study by following the element references, an XPath expression is constructed by analyzing the `OID` attributes resulting in a path as shown in Listing 4. That expression (`$destination` in Listing 3) is used as reference attribute within the XForms input element. Note again that this expression used in the presentation layer is totally compatible with the data model used in the database.

```
ClinicalData[@StudyOID = "ST.1" and
@MetaDataVersionOID="MD.1"]/SubjectData[@SubjectKey="Subject1"]
/Study[@StudyOID="SE.1"]/FormData[@FormOID="FO.1"]
/ItemGroupData[@OID="IG.1"]/ItemData[@ItemOID="IT.1"]
```

Listing 4: XPath generated by traversing ODM Study element to access ItemData.

4.3 Details of the Implementation

Several implementations of XML databases and XForms interpreters exist. To allow a broad adoption of the x4T system by clinical sites, it is necessary to keep the per-instance costs low. Thus, we have searched for free software implementations that fulfill our requirements concerning sufficient functions and stability. In an early project phase, we found BaseX[1] and eXist[2], which are free XML databases and suitable in general.

XForms interpreter have to be classified into server and client side implementations. As our scenario does not allow the installation of additional software on the client side, only server-side implementations are relevant here. There are two major free implementations of those, Orbeon[3] and betterFORM[4].

The eXist database is distributed with an integrated betterFORM implementation and developers of both cooperate. Both applications are developed and maintained by companies offering commercial support. They also have an active community discussing on mailing lists. The integration of betterFORM and eXist has been the core driver to prefer these implementations over the other ones mentioned.

Figure 4 shows a form generated by betterFORM in a Web browser. It is based on XForms that is generated by XSLT from the ODM definition.

The described software stack has been implemented and is currently used for setting up a dermatology database that is expected to comprise thousands of patients. Though it is currently too early for any form of final judgement, preliminary experiences look promising: The system replaces an Excel file that was used to store data that was manually extracted from the source system. The ODM-based approach and the XML technology-based generation of data structure and presentation layer now allow for a

1 http://basex.org/

2 http://exist-db.org

3 http://www.orbeon.com/

4 http://www.betterform.de/en/index.html

quick adaption on setup. At runtime, there are indications for an enormous benefit concerning the staff's time used for documentation. Technically, the study setup is facilitated by the fact that the study definition solely depends on the ODM file and adoptions can be done by changing that.

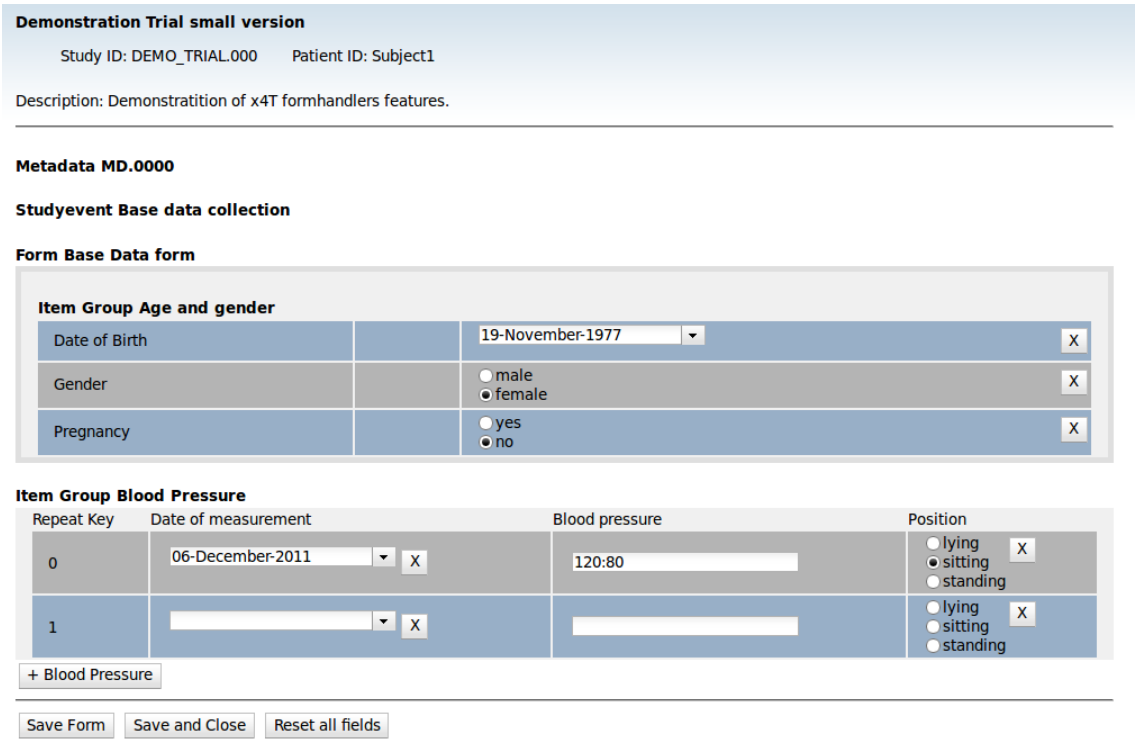

Figure 4: Form generated by betterFORM based on XForms standard.

5 Conclusion and Future Work

We have designed and implemented an architecture for generating forms in the clinical trials domain based on an XML definition. We are currently exploring the usability, performance, and scalability for studies under live conditions comprising up to thousand patients.

It has turned out to be beneficial to be able to generate studies out of the ODM definitions for which a variety of graphical editors is available[5]. We have found that clinical staff in research driven studies often uses Excel-based spreadsheets that are manually implemented and hard coded for each study. The setup time for studies is reduced considerably by automatic creation as described in this paper, and is immensely simplified by the XML stack. The ability to generate forms even attracted the interest of physicians who are not interested in the Single Source approach in general. The focus of our research has not primarily been on software architecture, but to handle a domain specific integration problem; the software architecture presented in this paper has turned out to be very helpful.

5 e.g. ODM Study Designer (http://www.xml4pharma.com/CDISC_Products/ODMDesigner.html) or STUDY COMPOSER (http://www.xclinical.com/en/study-composer)

Prospectively an assisted process execution of studies is preferable in order to ensure that the actual study activities comply with the study protocol. There is an addition to ODM called SDM [CDISC, 2010b] that defines how study execution information can be embedded into ODM. As the SDM extension is quite new, we are not aware of specific modeling tools. The generic approach of business process modeling as described by [Schönthaler et al., 2011] is based on Petri nets and their extension into XML nets, and on tool support through the Horus Business Modeler; our plan is to exploit both in the clinical research domain. As Horus supports XML nets, we expect it to fit well into the XML architecture stack.

Acknowledgements

The authors would like to thank the anonymous reviewers for their constructive comments. The work of Christian Forster was supported by Deutsche Forschungsgemeinschaft (DFG) under grant DU 352/5-1 AOBJ: 570946.

References

[Berglund et al., 2010] Berglund, A., Boag, S., Chamberlin, D., Fernández, M. F., Kay, M., Robie, J., and Siméon, J. (2010). XML Path Language (XPath) 2.0. W3C recommendation, W3C. http://www.w3.org/TR/2010/REC-xpath20-20101214/.

[Bhattacharya and Neamtiu, 2011] Bhattacharya, P. and Neamtiu, I. (2011). Assessing Programming Language Impact on Development and Maintenance: A Study on C and C++. In *Proceeding of the 33rd International Conference on Software Engineering*, ICSE '11, pages 171–180, New York, NY, USA. ACM.

[Boag et al., 2010] Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J., and Siméon, J. (2010). XQuery 1.0: An XML Query Language (Second Edition). W3C Recommendation, W3C. http://www.w3.org/TR/2010/REC-xquery-20101214/.

[Boyer, 2009] Boyer, J. M. (2009). XForms 1.1. W3C recommendation, W3C. http://-www.w3.org/TR/2009/REC-xforms-20091020/.

[Bray et al., 2008] Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (2008). Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C recommendation, W3C. http://www.w3.org/TR/2008/REC-xml-20081126/.

[Cagle, 2008] Cagle, K. (2008). Metaphorical Web and XRX. *O'REILLY Community*. http://broadcast.oreilly.com/2008/09/metaphorical-web-and-xrx.html.

[CDISC, 2010a] CDISC (2010a). Operational Data Model. Technical Report 1.3.1, Clinical Data Interchange Standards Consortium.

[CDISC, 2010b] CDISC (2010b). Study Design Model in XML. Norm 1.0, Clinical Data Interchange Standards Consortium.

[Davis and Maguire, 2011] Davis, C. and Maguire, T. (2011). Xml technologies for restful services development. In *Proceedings of the Second International Workshop on RESTful Design*, WS-REST '11, pages 26–32, New York, NY, USA. ACM.

[Dugas et al., 2009] Dugas, M., Breil, B., Thiemann, V., Lechtenbörger, J., and Vossen, G. (2009). Single source information systems to connect patient care and clinical research. *Stud Health Technol Inform*, 150:61–65.

[Dziuballe et al., 2011] Dziuballe, P., Forster, C., Breil, B., Thiemann, V., Fritz, F., Lechtenbörger, J., Vossen, G., and Dugas, M. (2011). The Single Source Architecture x4T to Connect Medical Documentation and Clinical Research. In *Proc. 23rd International Congress of the European Federation for Medical Informatics (MIE 2011)*.

[Fowler, 2003] Fowler, M. (2003). *Patterns of enterprise application architecture*. The Addison-Wesley signature series. Addison-Wesley.

[IHE International, 2009] IHE International (2009). Ihe iti technical framework supplement retrieve form for data capture (rfd).

[Kaufmann and Kossmann, 2009] Kaufmann, M. and Kossmann, D. (2009). Developing an Enterprise Web Application in XQuery. In *Proceedings of the 9th International Conference on Web Engineering*, ICWE '9, pages 465–468, Berlin, Heidelberg. Springer-Verlag.

[Kay, 2007] Kay, M. (2007). XSL Transformations (XSLT) Version 2.0. W3C Recommendation, W3C. http://www.w3.org/TR/2007/REC-xslt20-20070123.

[Kepser, 2004] Kepser, S. (2004). A Simple Proof of the Turing-Completeness of XSLT and XQuery. In Usdi, T., editor, *Extreme Markup Languages 2004*.

[McCreary, 2007] McCreary, D. (2007). Introducing the XRX Architecture: XForms/REST/XQuery. *Dr. Data Dictionary*. http://datadictionary.blogspot.com/2007/-12/introducing-xrx-architecture.html.

[Myrtveit and Stensrud, 2008] Myrtveit, I. and Stensrud, E. (2008). An empirical study of software development productivity in C and C++. In *Norsk informatikkonferanse, NIK 2008*.

[Robie et al., 2011] Robie, J., Chamberlin, D., Dyck, M., Florescu, D., Melton, J., and Siméon, J. (2011). XQuery Update Facility 1.0. W3C Recommendation, W3C. http://-www.w3.org/TR/2011/REC-xquery-update-10-20110317/.

[Schönthaler et al., 2011] Schönthaler, F., Vossen, G., Oberweis, A., and Karle, T. (2011). *Geschäftsprozesse für Business Communities — Modellierungssprachen, Methoden, Werkzeuge*. Oldenbourg.

[Sperberg-McQueen, 2005] Sperberg-McQueen, C. M. (2005). Xml. *Queue*, 3:34–41.

[Sun Microsystems, 2001] Sun Microsystems (2001). Core J2EE Patterns - Data Access Object. http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html.

[Tatarinov et al., 2002] Tatarinov, I., Viglas, S. D., Beyer, K., Shanmugasundaram, J., Shekita, E., and Zhang, C. (2002). Storing and querying ordered xml using a relational database system. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 204–215, New York, NY, USA. ACM.

[Wikibooks, 2010] Wikibooks (2010). Web Development with XRX. http://-en.wikibooks.org/wiki/XRX.