available at www.sciencedirect.com

ScienceDirect

www.elsevier.com/locate/ecolinf

A metadata-driven framework for generating field data entry interfaces in ecology

Christopher Jones[b,*], *Carol Blanchette*[b], *Matthew Brooke*[a], *John Harris*[a], *Matthew Jones*[a], *Mark Schildhauer*[a]

[a]*National Center for Ecological Analysis and Synthesis, University of California, Santa Barbara, United States*
[b]*Marine Science Institute, University of California, Santa Barbara, United States*

ARTICLE INFO

Article history:
Received 2 November 2006
Accepted 24 June 2007

Keywords:
Ecological data entry
User-interface generation
Metadata
Handheld computer
Schema inference

ABSTRACT

With the advent of affordable yet powerful handheld computers, many ecologists now capture data electronically in the field, while avoiding transcription errors found in more traditional field notebook or slate-based approaches of recording data. Ecologists either use simple forms from generic handheld software that doesn't allow for needed optimizations, or else they must spend substantial effort in the design and optimization of handheld software user interfaces. These efforts often produce highly customized software that is not easily repurposed, and that produce non-standard, undocumented data formats.

Our research is focused on reducing this effort by providing a flexible framework that uses highly structured metadata to drive the generation of customized data entry interfaces. Our approach relies on formalized metadata encoded in either XML Schema or XML instance documents, which facilitate the creation of graphical interfaces from arbitrary data schemas. Specifically, we leverage the rich metadata descriptions and structures found in the Ecological Metadata Language to create data entry forms, and output data sets that adhere to this specification. The interfaces are refined by using data-typing and domain information found in the structured metadata to generate validation routines that assist in maintaining data integrity during the entry process. The framework utilizes a rule-based engine to promote both reuse and extensibility of the modules that are created for any given data entry effort. Interfaces are rendered using open, Internet-based standards to maintain portability and to advocate the sharing of software components.

The framework represents an initial implementation of a customizable user interface generator, which should have broad applicability to researchers in need of rapidly creating field and laboratory-based forms for varied collection efforts.

1. Introduction

Ecologists are increasingly called upon to address broad environmental issues such as climate change and loss of biodiversity, and in doing so are expanding both the temporal and spatial scales at which they collect data. Field experiments and monitoring efforts play a crucial role in acquiring the critical information needed to analyze the patterns that are observed and to understand the processes involved in a changing environment. Although most ecological data are collected by individual researchers working in relative isolation on highly focused topics, growth in the development of collaborative, multi-disciplinary research programs has heightened the need to efficiently and comprehensively collect, document, communicate, and ultimately preserve primary research data.

* *Corresponding author.*
E-mail address: cjones@msi.ucsb.edu (C. Jones).

1574-9541/$ – see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.ecoinf.2007.06.005

Many ecologists conduct field experiments and monitoring efforts using traditional data entry approaches that utilize field notebooks, or, in the case of wet or underwater environments, hand-written observations on waterproof slates. Due to these relatively free-form methods of data entry, data-typing and formats are generally unstructured, and input values are unconstrained. Inconsistent or badly structured data are often an unintended consequence. Moreover, this process introduces another opportunity for error during the transcription itself. Important metadata such as coordinates, sampling design descriptions, step by step collection procedures, and variable meanings, types, and units are often kept isolated from data tables because they do not conform to the regular structure of a spreadsheet. This information may be crucial for correct data interpretation and analysis. Because broad synthetic research efforts rarely review fine details about the raw datasets, errors or biases that originate at this stage tend to persist in subsequent research.

In order to address some of issues above, ecologists have employed electronic data collection techniques using consumer-level handheld computers such as PalmPilots™ and PocketPCs™. Off-the-shelf software available for these platforms allow for a reasonable degree of flexibility in producing data entry applications and for interfacing them with external hardware additions such as barcode scanners and GPS receivers. However, many ecological experiments and monitoring efforts require a highly customized user interface (UI) that is tailored to the collection methods derived from the sampling design (Van Tamalen, 2004). Efficiency can be a critical factor due to time constraints imposed on the collectors (e.g. rising tides in intertidal ecology), and in-the-field configurability is often needed to deal with unforeseen data entry issues (e.g. assigning temporary taxonomic codes to unidentifiable taxa in the field for later keying in the laboratory.) Given these and other demands, the creation of highly usable data entry applications becomes a non-trivial task, often requiring a dedicated programmer to develop and test multiple iterations of an application, with each research group's development effort often starting from scratch.

The Jalama Project (Gaines et al., 2001) is an open source software development effort aimed at improving ecological informatics by providing a framework that allows for the creation of customizable and reusable data entry interfaces derived from highly structured and rich metadata descriptions of arbitrary data schemas. In turn, the software produced from the Jalama Project is compatible with a larger ongoing effort to archive and distribute ecological metadata and data via the Knowledge Network for Biocomplexity (KNB) (Jones et al., 2001; Reichman et al., 1999; Berkley et al., 2001). One of the main products produced from the KNB effort is the Ecological Metadata Language (EML) (Nottrott et al., 1999), a community-driven content specification designed to thoroughly document ecological resources as described in Michener et al. (1997), in a robust, extensible, and machine-readable format. The instance documents of the EML language are encoded in XML syntax (Bray et al., 1998) and are validated against the content and structures stipulated in a series of modular XML Schema (Biron and Malhotra, 2004) documents which implement the content specification of EML.

In this paper, we offer an approach that utilizes meta-information from both XML instance documents and XML Schema documents (such as EML documents) to generate graphical user interfaces. The distinctive feature of this approach is the automated generation of data entry forms without a priori knowledge of the data schema of the target dataset. In Section 2, we introduce the design and motivation behind the Jalama user interface generator system, and discuss the meta-information needed from input components for user interface (UI) creation. In Section 3, we describe the features of the Mozilla application framework that allow us to express the UI behavior, presentation, and application flow. We also introduce a rule-based approach to user interface pattern matching, field population, and application flow. We conclude in Section 4 by discussing some of the future work needed to refine the UI generation framework.

2. Leveraging meta-information

The basis of an intuitive and successful user interface comes from the intelligent presentation of the underlying data model derived from the procedural methods in the sampling design of a given collection effort or experiment. It is therefore common practice to customize a given data or metadata entry application based on the logical schema of a data model from a specific physical model, for example tabular ASCII files or relational database tables. The flow of the application is often hard-coded based on the particular needs of the collection effort. The customizations required are often tightly linked to the software system chosen to create the application, and therefore re-purposing the software becomes a non-trivial task. The consequences of this type of software development model include large up-front efforts of dedicated time, and often re-implementation of the same basic software framework for each collection effort, especially across uncoordinated institutional efforts. Therefore, in order to reduce the effort needed for each application development endeavor, we concentrate on exploiting the metadata associated with common data models, both at the abstract schema level and the instance level. In the initial implementation of the framework, we focus on deriving critical meta-information from the Ecological Metadata Language (EML) schema and instance documents and encode collected data from the generated user interfaces back into EML. By doing so, we avoid the situation where critical metadata are physically divorced from the data entities produced from a given collection effort, increasing their longevity and applicability to future synthetic analyses.

2.1. *Main workflow of the Jalama framework*

Employing the Jalama framework consists of a four sequential steps, depicted in Fig. 1. These steps include: 1) developing an EML document that describes the data to be collected, 2) submitting this EML document as input to the UI generation software, 3) syncing the resultant user interface to a field computer running the client software that will render and run the collection application, and 4) syncing the newly-created EML documents and data that were collected in the field back to the desktop computer for archival. The first step is typically

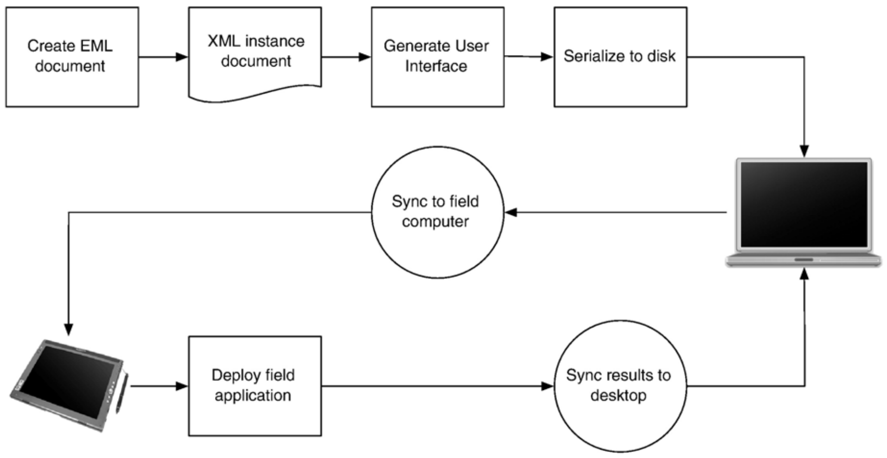


Fig. 1 – Overview of the workflow involved in generating, deploying, and returning data from form-based user interfaces for ecological data collection.

accomplished through the use of the ecological data management client called Morpho (Higgins et al., 2002), whereby a scientist uses a form-based wizard to describe the details of the dataset, including high-level discovery metadata such as title, creators, and abstract, as well as low level variable details including definitions, associated units, domain constraints on allowable entry values, and other detailed data-typing information. The second step of generating the user interface definition is likewise typically performed from within Morpho. We have extended the Morpho software application with a Jalama-based plugin that allows a researcher to choose the target EML data package and invoke the UI generation engine via a menu command. The synchronization steps are initiated on either the desktop computer running Morpho, or on a field computer running the Jalama Client software, whereby the serialized user interface file and the original EML metadata file are transferred via HTTP using web server software embedded in the application.

2.2. *Instance-specific meta-information*

In this framework, we have chosen to generate user interfaces from the information found in XML Schema-based metadata documents, specifically those expressed in the Ecological Metadata Language. Although the framework may take an arbitrary XML syntax and produce a generic user interface, we capitalize on the detailed structures and constraints present in EML documents that are highly pertinent to the scientific data models being described in the EML. Listing 1 shows some of the pertinent components of an EML document.

```
<eml …>
  <dataset>…
    <dataTable>
      <entityName>
        Coastal Biodiversity Survey Table
      </entityName>…
      <attributeList>
        <attribute>
          <attributeName>position</attributeName>
          <attributeLabel>
            transect position
          </attributeLabel>
          <attributeDefinition>
            The position along the transect in
            meters.
          </attributeDefinition>
          <measurementScale>
            <ratio>
              <unit>
                <standardUnit>
                  meter
                </standardUnit>
              </unit>
              <precision>0.05</precision>
              <numericDomain>
                <numberType>real</numberType>
              </numericDomain>
            </ratio>
          </measurementScale>
        </attribute>
        <attribute>
          <attributeName>
            class_code1
          </attributeName>
          <attributeLabel>
            Classification Code 1
          </attributeLabel>
          <attributeDefinition>
            The first observed classification at a
            point along the transect.
          </attributeDefinition>
          <storageType>text</storageType>
          <measurementScale>
            <nominal>
              <nonNumericDomain>
                <enumeratedDomain enforced="yes">
                  <codeDefinition>
```

```
                    <code>ACASPP</code>
                    <definition>
                       acanthina spp.
                    </definition>
                 </codeDefinition>
                 <codeDefinition>
                    <code>ACRSPP</code>
                    <definition>
                       acrosiphonia spp.
                    </definition>
                    …
                 </codeDefinition>
              </enumeratedDomain>
           </nonNumericDomain>
        </nominal>
     </measurementScale>
     <missingValueCode>
        <code>.</code>
        <codeExplanation>
           A period (.) represents any missing
           value for this column.
        </codeExplanation>
     </missingValueCode>
   </attribute>…
  </attributeList>…
 </dataTable>
</dataset>
</eml>
```

Listing 1. Detailed data schema information for variables in a dataset described in the Ecological Metadata Language.

In this example EML document, a dataset contains a single data table with two variables described in detail. The first variable, “position”, is a measure of distance along a fixed transect. The metadata record contains highly pertinent information that will drive the creation of the data entry forms. Although the name of this variable is readable, there is also an “attributeLabel” element that provides a more human-readable label that would be used in marking an input field on the data entry form. Likewise, the “position” variable is constrained to being a *real* number in the numeric domain, measured in meters with a precision of 0.05 m. This information is extracted by the UI generation software in order to modify both the type of input widget presented to the researcher and the behavior of the associated widget by customizing the validation routines that are fired upon data entry. For instance, in this case, if a character string were entered into the input field, upon validation, the researcher would be visually warned that the value falls outside of the bounds of the allowable values, and would be prompted to correct the entry.

Likewise, the second attribute in Listing 1 named “class_code1” reveals that the variable's data type falls in the non-numeric, text domain, is constrained by an enumerated list of coded values, and allows missing values to be represented with a period (.). Again, the generation software takes advantage of these metadata in order to produce an appropriate data widget for the variable. In this case, for example, a drop down list constrains the values to those in the coded list.

2.3. Schema-level meta-information

Despite the explicit details regarding data typing, validation, and constraints found in an EML instance document, meta-information that governs the structure and content of the instance document itself is also critical to the automated generation of corresponding user interfaces. The W3C XML Schema Recommendation provides an abstract data model that can be used to assess the validity of an XML document that adheres to the XML Infoset specification (Cowan and Tobin, 2004). The abstract components defined in the recommendation include type definitions and element declarations, among other components, that govern the structure of XML instance documents. The fundamental building blocks used in a set of XML Schema documents can be examined in order to extract the properties that affect user interface behavior and flow. For instance, XML Schema content model groups include “sequence”, “choice”, or “all” elements, which specify the sequential (order matters), disjunctive (only one), or conjunctive (all possible) structures available in an XML instance document that adheres to the schema. Therefore, a complex type class definition in the EML schema specification that shows a “choice” element containing three separate sub elements would prompt the UI generation software to present the user with a “choice” widget such as a radio button control.

Likewise, each element within the EML schema is governed by the cardinality constraints expressed in the element definition. As seen in Listing 2, both the minimum and maximum allowable occurrences of the element have an impact on the widgets rendered in the user interface. For instance, a form may be generated that allows a researcher to list the personnel associated with a data collection effort. In the EML schema, the “personnel” element declaration is optional and repeatable with no upper limit on the number of possible elements in the document. This type of information is crucial to user interface generation, since a listbox-type widget would need to be rendered in order to accommodate one to many “personnel” instances and their child elements.

```
<xs:element name="personnel"
  minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:complexContent>
      <xs:extension base="rp:ResponsibleParty">
        <xs:sequence maxOccurs="unbounded">
          <xs:element name="role" type="rp:RoleType">
          </xs:element>
        </xs:sequence>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Listing 2. An example element declaration in the Ecological Metadata Language showing cardinality constraints and extension from a base type.

2.4. Schema inference

As was mentioned earlier, the Jalama framework affords a certain amount of flexibility when ingesting meta-information used to drive the generation of data entry forms. This framework is extensible in that it will accept arbitrary, well-formed XML instance documents in addition to any grammar expressed as an XML Schema document. In the latter scenario, the XML Schema document is evaluated for element and attribute content declarations, and a minimum-required XML instance document is produced for reference during the UI generation steps. In the former scenario, the Jalama framework utilizes the Castor Project's (Exolab Group, 2004) Java-to-XML binding libraries to infer either an explicit or a general schema based on the elements and attributes present in the instance document. For instance, an explicit schema for a sequence of three "email" elements found in the instance document would assume that there is a one-to-one relationship with the elements and their declarations produced in the resultant schema document, with each declaration's maxOccurs attribute set to "1", and each element receiving a unique name (<email. 1>, <email. 2>, <email. 3>). On the other hand, a general schema may be inferred where in the same scenario, a single, repeatable <email> element declaration is produced with the maxOccurs attribute set to "3".

As is depicted in the initial steps of Fig. 1, we are able to produce the inputs required for subsequent UI generation steps by gathering meta-information from instance documents and their associated XML schemas. This process ultimately produces two Java objects, a Document Object Model (DOM) of the well-formed but non-validated instance document, and a Post SchemaValidation Infoset (PSVI) object of the validated instance document. These objects provide consistent programming interfaces that are called during the UI generation workflow. The latter refers to the augmented XML Infoset that results from processing the XML instance document conforming to its associated schema.

3. User interface generation

Once the DOM and PSVI models are available for evaluation, two activities must be undertaken to generate an intuitive and functional user interface without a priori knowledge of the data schema: 1) Apply a series of progressively-detailed, configurable business rules that refine the metadata-derived user interface in terms of its structure, content, logic, and application flow and, 2) Express the generated user interface in a portable syntax that can be rendered at run-time in a cross-platform application framework. We have chosen a rule-based technology called Drools (see http://legacy.drools.codehaus.org) that allows us to address (1) and separate the configurable components of the system from the core application logic. Secondly, we have chosen the Mozilla application framework (see http://www.mozilla.org) as the means of expressing, rendering, and running the data entry form applications generated from the system. Mozilla's XML User Interface Language (XUL) and XML Binding Language (XBL) provide the core technologies that allow us to render cross-platform data entry applications that leverage existing Web-based programming standards. As depicted in Fig. 2, the Business Rules Engine (BRE) is applied in three instances: to match logical patterns in the PSVI model to user interface components, to populate and constrain those components with instance-specific data, and to determine the physical layout that is optimal for the collection protocol and the properties of the collection device. Execution of the UI generation workflow results in the creation of a XUL-based interface definition represented in a serialized Java object that is written to disk along with the original EML metadata document that was used to initially drive the process.

3.1. *Mozilla application framework*

The Jalama software builds upon the mature and extensible application development framework of the Mozilla project. We chose to adopt the Mozilla framework because of a number of reasons, including:

- Mozilla's support of Internet-based standards such as Cascading Style Sheets (CSS), the Document Object Model (DOM), the Resource Description Framework (RDF), and XML
- The ability to express user interfaces in a transparent, declarative XML-based syntax (XUL) that can be extended using a straight-forward binding language (XBL) and a well-known scripting language (ECMAScript)
- The ability to render XUL documents as user interfaces immediately, without the need for compilation or pre-processing

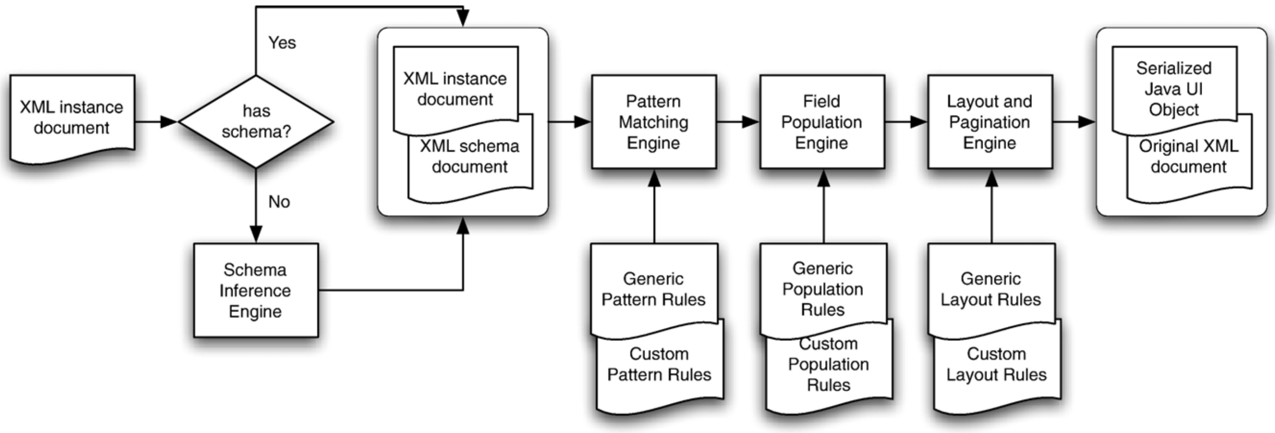


Fig. 2 – Overview of the Jalama user interface generation architecture.

- The open source nature of the project
- Mozilla's widespread adoption and large user community.

Although we chose to adopt the Mozilla Layout Engine as the target rendering engine for the initial implementation of the Jalama software, the framework is not irrevocably tied to using Mozilla since the Jalama architecture has been developed to provide a flexible, plugin-based approach to adding or removing rendering engines, so that other technologies could be used to replace or complement Mozilla in future. For example, a code generation module could be used to create user interfaces from the same XUL documents that have been rendered by Mozilla.

3.2. *Form library*

By adopting the XUL and XBL languages, we benefit from the existing library of fundamental user interface widgets provided within the Mozilla framework, such as <button>, <textbox>, <listbox>, <scrollbox>, etc. However, for generating customized forms that meet the needs of arbitrary data collection efforts, we needed to build a form library that was accessible to the Pattern Matching Engine in order to provide a rich set of modular, visual components more tailored to scientific data entry in the field. The Jalama software includes a flexible architecture of XBL bindings — modular components whose visual characteristics are defined by the XUL markup language, and whose behavior is defined using ECMAScript (see http://www.w3.org/TR/xbl/). Each of the bindings implements a common Application Programming Interface (API) to ensure consistency of behavior across the collection of visual components. Listing 3 shows an example XBL binding (shortened for readability) that defines a "date_textbox" input control in the XUL language. This visual component extends the "base_textbox" component, and therefore inherits the methods and behavior of its parent class. However, it overrides the validate () method by providing its own ECMAScript code that validates the user's input against the date–time format that is specified in the variable definition in the EML instance document that this widget is being used to represent. If the input does not adhere to the specified format, the researcher is visually prompted to correct the entry in the field.

```
<binding id="date_textbox"
   extends="#base_textbox">
   <implementation>
      <method name="validate">
         <body>
            <![ CDATA[
               var input= this.value;
               if (this.dateTimeFormat!=null
                  && false=lib.dateTimeMatchesFormat (
                  input, this.dateTimeFormat)) {
                  if (lib.isDateTime(input)) {
                     input = lib.formatDateTime(
                        input, this.dateTimeFormat);
                        if (input!=null) this.value=input;
                        else return this._doHilite();
                  }  else{
                  return this._doHilite();
                  }
                  }  else{
                     this.setHilite(false);
                     return true;
                  }
               ]]>
            </body>
         </method>...
   </implementation>
</binding>
```

Listing 3. An example XBL binding written in XUL and ECMAScript.

3.3. *Business rules engine*

The Jalama framework adopts a rule-based approach to customizing data entry forms for field applications. This approach allows for an easily modifiable UI generation configuration that could be adapted by a data manager familiar with declarative programming and is acquainted with procedural programming. By utilizing a Business Rules Engine such as Drools, the Jalama UI generation logic system can be extended by "chaining" any number of "rules". This "rule-chaining" concept promotes an extremely configurable UI generation application and promotes the reuse of rules where applicable. The Jalama system refines the rule-based approach by defining and applying a generic, coarse grained set of rules that can be subsequently overridden by a customizable set of rules. This type of architecture allows organizations deploying the Jalama software to develop their own UI rule configurations, but to rely on the generic rules provided within the framework as a fallback. The Drools BRE is an open source project that has an active developer community and makes use of Java and XML technologies for its configuration files.

Currently the Jalama framework uses the Drools BRE to store and process rules for the three main components of the application: UI widget pattern matching, UI widget population, and the UI layout modules. We have manually configured the business rules configuration files for each of these scenarios. However, we expect that as the framework matures, a more graphical system would be developed that interviews a researcher for information regarding measurement methodology and experimental procedures, and uses the responses to generate rules that dictate the layout and logic of their forms for their particular collection effort.

3.4. *Pattern matching*

To demonstrate the simplicity of the rule-based approach, Listing 4 continues the "date–time" example of the XBL widget described earlier. In the "match_date" rule definition, the Pattern Matching Engine will evaluate the element declaration passed in from the PSVI model, and compare the element declaration's structural type to the "simple date" type found in the XML Schema possible types. If the condition is a match, then the consequence of the rule is to set the widget type for this element to be the XBL binding defined by the "date_textbox" identifier. Following this example, each fragment of the PSVI model is passed to the Pattern Matching Engine for evaluation based on the chain of business rules defined in the configuration files. Each component of the data schema is then either mapped to a visible UI widget that will be rendered in the data entry application, or it is mapped to an "invisible" container element used to organize the layout of the form using Cascading Style Sheet presentation properties. The Pattern Matching Engine produces an ordered map of XPath (Clark and DeRose, 1999) statements that point to each node of the EML document, along with the corresponding XBL binding assigned to the node.

```
<rule-set name="jalama_uigeneration" …>
  <rule name="match_date">
    <parameteridentifier="element">
      <java:class>
        edu.ucsb.nceas.jalama.generation.rule-
        sengine.\
        XSElementDeclarationWrapper
      </java:class>
    </parameter>
    <java:condition>
      element.getType () ==
      edu.ucsb.nceas.jalama.generation.\
      XMLSchemaElementTypes.SIMPLE_DATE
    </java:condition>
    <java:consequence>
      <![ CDATA[
        element.setWidgetType(
        "textbox_bindings.xml#date_textbox");
        modifyObject(element);
        retractObject(element);
      ]]>
    </java:consequence>
  </rule>…
</rule-set>
```

Listing 4. An example of a business rule for the Jalama UI generation system.

3.5. *User interface field population*

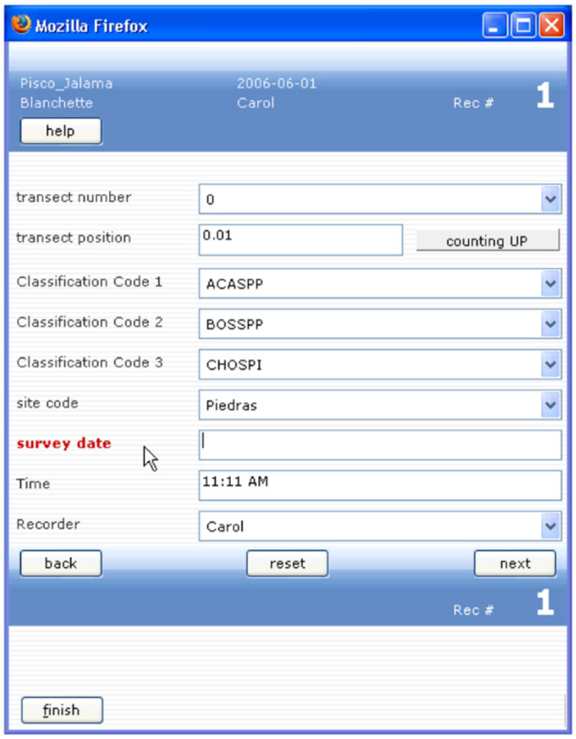
Once the pattern matching step has returned the XPath and XBL Binding ordered map, the map is further refined by the Field Population Engine. In this processing step, the empty user interface widgets that were chosen by the Pattern Matching Engine gain constraints that are defined in the original PSVI model. Again, a series of both generic and customizable business rules drive the process, and each XPath-XBL binding pair in the ordered map are evaluated. A typical example of field population would involve an enumerated list of values to be presented to the researcher as choices. For instance, Listing 5 shows the XML fragment (shortened for readability) of an EML instance document that represents the dataset variable located at the XPath location of /eml/dataset/dataTable[1]/attributeList[1]/attribute [6]. The Field Population Engine evaluates this location within the PSVI model, and based on it's structural type expressed in the EML, modifies the XBL binding with the enumerated values in the instance data. In this case, a <selectbox> has been assigned by the Pattern Matching Engine, which in turn is populated with <menuitem> elements that contain the EML coded strings as values ("Piedras", "Lompoc", etc.) that are presented in the drop-down widget.

```
<attribute>
  <attributeName>
    site
  </attributeName>
  <attributeLabel>
    site code
  </attributeLabel>
  <attributeDefinition>
    The code that represents the
    sampling location.
  </attributeDefinition>
  <measurementScale>
    <nominal>
      <nonNumericDomain>
        <enumeratedDomain enforced="yes">
          <codeDefinition>
            <code>Piedras</code>
            <definition>
              Piedras Blancas Lighthouse, CA
            </definition>
          </codeDefinition>
          <codeDefinition>
            <code>Lompoc</code>
            <definition>
              Lompoc Landing, CA
            </definition>
          </codeDefinition>
          …
        </enumeratedDomain>
      </nonNumericDomain>
    </nominal>
  </measurementScale>
</attribute>
```

Listing 5. An example of an enumerated list found in EML used to populate a drop-down UI widget.

3.6. *Layout and pagination*

Given the fully populated XPath and XBL Binding map, the final step is to determine the overall flow of the application based on patterns and groupings present in the data schema within the PSVI model, and by evaluating the step-by-step procedures involved in the sampling design metadata. The Layout and Pagination Engine is also driven by a set of generic and customized business rules. The final product of this step is not only a single XUL-based UI description file, but rather an array of XUL files that are used in coordination to model the step-by-step data entry procedures of the collection effort. Thus far, the Jalama framework has implemented a generic "record per screen" layout that is typical of repetitious field data collection present in monitoring programs. Fig. 3 depicts a rendered user interface on a TabletPC field computer, and highlights a number of the UI components that have been selected to represent the variables found in the original EML-described dataset. In this implementation, high-level metadata that pertain to the entire collection effort are rendered at the top off the screen, such as the data table name, the date of this collection series, the collector's name, and which record is currently being modified. Each modifiable variable is arranged in the main



Mozilla Firefox

Pisco_Jalama 2006-06-01
Blanchette Carol Rec # 1

transect number	0
transect position	0.01 counting UP
Classification Code 1	ACASPP
Classification Code 2	BOSSPP
Classification Code 3	CHOSPI
site code	Piedras
survey date	
Time	11:11 AM
Recorder	Carol

Rec # 1

Fig. 3 – A customized, rendered user interface generated from EML meta-information.

content area of the window. Each variable presented includes the value of the <attributeLabel> element from the original EML document on the left side of the interface, and a corresponding populated data entry widget on the right side of the screen. In the case of the "transect position" widget, an auto-incrementing numeric textbox widget has been rendered based on the custom business rules for pattern matching. Likewise, the "survey date" label is displayed in red, demonstrating the result of calling the validate ( ) method on the custom <date_textbox> XBL widget when the field is left blank.

The Layout and Pagination UI generation step is critical to the customization of data entry applications, since it heavily influences the cadence of the collection effort. In some cases, a non-optimized application flow is sufficient for successful data collection, however, most field-based researchers tend to be time-limited during the data collection process and are interested in increasing their efficiency in the field. Likewise, layout and pagination issues in electronic field collection can affect data quality as well. Without being able to "go back" or "go forward" through screens or presented questions, incorrect entries would need to be noted and would be propagated into the final dataset produced by the forms. Although the responsiveness of each widget to invalid data-types certainly helps to maintain record-by-record integrity, mistakes due to forgetfulness or errors in judgement can only be addressed by a flexible application flow.

4. Conclusions

Utilization of graphical user interface generation techniques can be a viable approach to producing responsive data entry applications for ecological field studies. As previously noted, scientists place a high value on optimized data entry forms in order to maximize the results of field-based observation programs. Leveraging meta-information found in highly structured and strongly-typed content specifications such as the Ecological Metadata Language can provide a large portion of the building blocks needed to produce customized user interfaces.

Likewise, using rule-based approaches to configuring application logic can greatly improve the quality and maintainability of user interface generation software, especially if they are developed using a community-based approach. Most data collection applications share very similar properties, and at times only diverge due programming language choice and platform restrictions. By capitalizing on the rich, open-source application programming framework developed by the Mozilla Foundation, we are able to deploy a full-fledged application based largely on declarative programming techniques, common web-scripting languages, and Internet-based standards.

By incorporating meta-information throughout the application development and deployment cycles, we avoid jeopardizing the longevity and usefulness of field-collected datasets. The UI generation and deployment framework we have presented represents an initial implementation of a round-trip application development system, from metadata creation, UI generation, application deployment, and creation of newly collected datasets that are fully described according to the details of the original metadata document. This open-source framework should have broad applicability to research groups in need of frequent form creation for varied data collection efforts.

4.1. *Future work*

Although we made the decision to use the Mozilla Layout Engine for the first rendering implementation of this project, we also see opportunities to take other rendering approaches. The application footprint of Mozilla, combined with a Java Virtual Machine for the initialization and synchronization portions of the software, becomes a significant factor in the deployment on low-memory handhelds and mobile phones. We envision a rendering component that performs code-generation to transform UI interfaces expressed in XUL and XBL into executable binary applications for wider deployment. Lastly, the pagination and layout component of the Jalama framework needs customization work in order to accommodate a wide variety of sampling regimes. For example, we envision a library of layout templates that correspond to known experimental designs in ecology such that a researcher could choose a template based on the design that has been applied to the experiment or monitoring effort. Properties of the template would be exposed as a graphical dialog that allows the researcher to further customize the flow of the application.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. DBI-0131178. We thank the National Center for Ecological Analysis and Synthesis and the Marine Science Institute at the University of California, Santa Barbara, for support during the development process.

REFERENCES

Berkley, C., Jones, M., Bojilova, J., Higgins, D., 2001. Metacat: a schema-independent XML database system. Proceedings of the Thirteenth International Conference on Scientific and Statistical Database Management, p. 0171.

Biron, P.V., Malhotra, A., 2004. XML schema part 2: datatypes. World Wide Web Consortium Recommendation 28 October 2004. http://www.w3.org/TR/xmlschema-2/(Available at).

Bray, T., Paoli, J., Sperberg-McQueen, C.M., 1998. Extensible markup language (XML) 1.0. http://www.w3.org/TR/REC-xml1998(February, Available at).

Clark, J., DeRose, S., 2004. XML Path Language (XPath) Version 1.0. World Wide Web Consortium Recommendation 16 November 1999. http://www.w3.org/TR/xpath(Available at).

Cowan, J., Tobin, R. 2004. XML Information Set (Second Edition). World Wide Web Consortium W3C Recommendation 4 February 2004. Available at: http://www.w3.org/TR/xml-infoset/.

Exolab Group, 2004. The Castor Project. http://castor.exolab.org/ (Available at).

Gaines, S.D., Jones, M.B., Schildhauer, M., Jones, C.S., Blanchette, C.A., 2001. Capturing data in the field: an application framework for easily creating custom data and metadata entry forms on handheld and desktop computers. National Science Foundation Award # DBI-0131178. (Available at http://jalama.ecoinformatics.org).

Higgins, D., Berkley, C., Jones, M., 2002. Managing heterogeneous ecological data using Morpho. Proceedings of the Fourteenth International Conference on Scientific and Statistical Database Management, pp. 69–76.

Jones, M.B., Berkley, C., Bojilova, J., Schildhauer, M., 2001. Managing scientific metadata. IEEE Internet Computing 5 (5), 59–68.

Michener, W., Brunt, J., Helly, J., Kirchner, T., Stafford, S., 1997. Nongeospatial metadata for the ecological sciences. Ecological Applications 7 (1), 330–342.

Nottrott, R., Jones, M.B., Schildhuaer, M.P., 1999. Using XML-structured metadata to automate quality assurance processing for ecological data. Proceedings of the Third IEEE Computer Society Metadata Conference, Bethesda, MD.

Reichman, O.J., Brunt, J., Helly, J., Jones, M.B., Willig, M.R., 1999. A knowledge network for biocomplexity: building and evaluating a metadata-based framework for integrating heterogeneous scientific data. National Science Foundation Award # DEB99-80154. (Available at: http://knb.ecoinformatics.org).

Van Tamalen, P.G., 2004. A comparison of obtaining field data using electronic and written methods. Fisheries Research 69, 123–130.