Acquiring and Delivering Lessons Learned for NASA Scientists and Engineers: A Dynamic Approach

Swapna Gupta  
The University of Alabama  
Department of Computer Science  
Box 870290  
Tuscaloosa, Al 35487  
sgupta@cs.ua.edu

J. Scott Hawker  
Rochester Institute of Technology  
Department of Software Engineering  
20 Lomb Memorial Drive  
Rochester, NY 14623  
hawker@se.rit.edu

Randy K. Smith  
The University of Alabama  
Department of Computer Science  
Box 870290  
Tuscaloosa, Al 35487  
rsmith@cs.ua.edu

ABSTRACT
Web forms are a common mechanism for collecting information online. They pose some limitations which negatively affect ease and flexibility of user interaction. These limitations are more pronounced when collecting information that cannot be broken down into well-defined and agreed upon structures. Lessons Learned or any kind of ‘knowledge’ is an example of such information. In this paper we introduce a concept of ‘Dynamic Forms’ which is aimed at enhancing user interaction while collecting unstructured information, ‘Lessons Learned’ in particular.

Categories and Subject Descriptors
H.5.2 [Information Interfaces and Presentation]: User Interfaces---Interaction styles

General Terms
Human Factors

Keywords
Web forms, Dynamic forms

1. INTRODUCTION
Organizations often collect information which can be pieces of ‘knowledge’ and store them for later use, creating a rich knowledge bank. One such facility at NASA is the Lessons Learned Information System (LLIS)[6]. LLIS provides storage and retrieval of lessons learned by and for NASA scientists and engineers while they perform their job related activities. It consists of a mechanism for lesson submission, a mechanism for approval of submitted lessons, a mechanism for scientists and engineers to search for lessons and a way of rendering the lessons to them. Figure 1 shows the system as a closed loop system facilitating production of Aerospace products and capabilities, which in turn generates new lessons for use.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

43rd ACM Southeast Conference, March 18-20, 2005, Kennesaw, GA, USA. Copyright 2004 ACM 1-58113-000-0/00/0004…$5.00.

First, static forms limit expressiveness. A lesson author may not find the appropriate field to record some piece of information in or the form may contain many
unnecessary fields. In absence of the appropriate field, the information would be forced into incorrect fields, which in turn can reduce effectiveness of the lesson-search mechanism. This is a disadvantage to the lesson submitters, users and the organization.

- Second, over time, these forms (lesson formats) tend to evolve. New formats are recommended and brought into practice. This requires changing the forms, database schemas or other storage formats, code to store/search the lessons and finally integrating the legacy formats into the new system.

3. RELATED WORK

Use of static web forms for submitting lessons has some disadvantages, as already noted in other papers [1, 3]. Different users have different needs. Many times a user needs to fill only a small number of fields for a given situation, rendering the other fields a mere distraction. In addition, we note that, sometimes, especially in a case of filling in information such as ‘knowledge’ (lessons), the semantics (from the lesson author’s point of view) of the available fields in the form are not aligned with the kind of information the user wants to enter. This leads to the problem of ‘lack’ of appropriate fields. The authors in [1] point out – “One may wonder, why is there still a need to verify the validity and logical design of forms, especially when the application’s designers already performed software requirements specification (SRS) on needs of their application?” And they answer “System developers tend to focus more on the application’s needs than users which may sometimes not realize actual user’s needs aspect and vice versa”.

This is corroborated by our experience with the LLIS and other lessons learned systems at NASA. Even though the forms are made after considerable amount of requirements analysis, they don’t satisfy all users’ needs. Apart from that, different centers at NASA have different formats for lesson submission. For example, the Orbital Space Plan (OSP) started out with the LLIS format, but later changed it to a format that suited its needs better, dropping some items and adding an item for identification of applicable disciplines, an item for categorization into specific OSP topics, and a general purpose item for “more information.” The NASA Technical Standards Program studied multiple formats to consolidate them into one common format that could be adopted across the agency, but ended up changing it when developing an Engineering Experience and Learning System prototype to integrate lessons learned and engineering training materials. They found a need for additional information, including a project or program associated with each lesson, and an identification of an engineering principle underlying each lesson learned. Thus, it is hard to successfully anticipate and rigidly define the “final, approved” agency-wide lessons learned format.

This leads to the second problem – evolution of forms and the consequences of that on the whole system. As the changes to the forms are suggested and implemented, new form formats are brought into place. This significantly affects the other components of LLIS, especially the search and rendering of lessons. Thus, there is a need for a quick way to incorporate changes to form formats and make them available for use without affecting or reducing effectiveness of other parts of the system, including availability of lessons already captured in other formats.

We have come across previous efforts in speeding up generation and modification of user interfaces. The authors in [3] introduce a Form Description Language (FDL) to define fields in an interface and relations between these fields. The authors designate the forms developed using FDL as ‘Dynamic Forms’. The system provides an automatic layout of fields, dynamically responding to the screen space available on user’s device. Also, it provides nested forms that collapse and expand a set of fields. This approach provides an efficient way of defining easily navigable forms and makes it easy for developers to quickly prototype new user interfaces, thus helping the requirements analysis process between users and developers. But, the fact remains that making changes to the interface requires a developer skilled in FDL.

The authors in [4] introduce the eXtensible Interface Markup Language (XIML). The idea is to have a common representation of interaction data eventually leading to standardization effort for representing interaction data. XIML is organized as a collection of interface elements which are grouped to form components. The component of our interest is the ‘Presentation’ component which defines the interaction elements to communicate with the users. An important point to note is that, the elements are defined at a level of abstraction such that the actual rendering of a specific interaction element can be left entirely to the target display system. This helps in multipurpose interface development (a common XIML specification rendered in Hypertext Markup Language (HTML) on a web browser and in Wireless Markup Language (WML) on a Personal Digital Assistant). The authors of the paper have developed XIML-to-HTML and XIML-to-WML converters. Our problem is more to do with flexibility in terms of semantics involved behind form fields and their relevance to the user and less to do with separating user interface specification from display devices. XIML doesn’t address our problem because this approach also requires an expert to make changes to the interface as well as to extend the language with more elements.

XForms [1, 7] have been introduced by the Worldwide Web Consortium (W3C) to overcome some of the limitations of HTML forms. They reduce their dependence on scripts to validate fields and provide functionalities that can save several round trips to the server. The submitted forms are interchanged as XML (eXtensible Markup Language) documents which act as ‘instances’ of the forms. XForms do not provide a solution to our problems. At best, we can move over to generating XForms instead of HTML forms in our system.

The authors in [2] describe an approach to model, represent and mark up ontologies. A study of generating web forms from ontologies marked up with presentation information is explained. This approach is closer to home with respect to
our problem. The paper separates roles of an ontology developer, forms developer and end user, which reflects the relative complexity of the tasks involved. Developing an ontology requires know-how of Object Role Modeling (ORM)[5]. A forms developer requires know-how of the ontology in order to identify the ‘fact types’ for which an input element is required on the form, taking care that it leads to no ambiguities. Also the forms developer must specify, among other things, the input form field that will go with the element. Lastly the end user needs to only fill in the forms, understanding the meaning of fields from their labels or by using links to their descriptions. We require that the process of element (‘entity’ in [2]) creation and form generation be sufficiently simple for them to be performed by the end users. Thus we have taken an approach similar to [2], but with an easier and simpler implementation and interface.

4. APPROACH AND UNIQUENESS
The proposed Dynamic Lessons Learned System (DLLS) would resolve the problems stated earlier. Figure 2 shows the DLLS model.

The Lesson submission process in DLLS takes place through Dynamic Forms. Please note the term Dynamic Form has been used in [3] also. The Dynamic Forms in [3] are created using FDL, while Dynamic Forms, in this paper, refer to the forms generated by users of LLIS while submitting their lessons. An email notification is sent to the discipline working group (approving authority) when a lesson is submitted. The discipline working group can edit the submitted lesson. They can approve or disapprove the lesson. If approved the lesson becomes available to scientists and engineers.

Typically, a form developer would glance over the library and pick the elements that sound semantically correct. Hovering the mouse over an element provides a brief description of the meaning associated with the element, clicking on an element would generate the corresponding input field on the form in the middle frame. If the form developers don’t find the element they are looking for, they will use the interaction in the right-hand frame to create the desired element. On submission, the new element is added to the library, and the lesson authors can then use it in their forms to submit their lessons. A new element when created, initially has a status of “unapproved”. This is reflected in the library by a thumbs-down icon. On submission of a new element, an administrator or approver is informed of it via email. The administrator or approving authority will study the element’s description and approve or disapprove it. A reason for disapproval could be redundancy, i.e., presence of a semantically equivalent element in the library. In case of disapproval, the system could be directed to move the content of this element in the submitted lesson to some existing element.

The Dynamic Form interface is shown in Figure 3. The left frame contains a library of elements. Each element has a meaning and an input form field associated with it. The middle frame is for showing the dynamically generated form. The right frame supports generating new elements.

A Dynamic Form is ‘dynamic’ because the interface allows a lesson submitter to assume the role of a form developer and generate a form which best suits his/her lesson. It is very well possible, that each submitted lesson is different in format from other lessons.

Figure 2. A Dynamic Lessons Learned Information System
Figure 3. User Interface for Lesson Submitter or Form Developer

The library of elements is stored in an XML file. A snippet of the XML file is shown in figure 4. For each element the following information is stored: name, a display name (which is used on the form), a type, a short description (shown when the mouse hovers on an element), a detailed description (shown when right-clicking the element), whether it has been approved (indicated by a thumbs up or thumbs down symbol) and a presentation style (this is not being used currently, but will be used later while rendering lessons). All of these, except the approval information, are provided by the form developer while creating the new element. Links are provided for lesson authors to help them understand the need for this information. The element type is used during form creation to dynamically generate the form fields. For example, a type String would be associated with a textbox. A type Large Text would be associated with a textarea. A type Static Text is used to add instructional information on the form. Types Choices(One/Many) and Choices(Many/Many) correspond to radio buttons or a drop down box and check boxes or a multiselect box respectively on the generated form. While creating new elements, choosing a type Choices, brings up fields for the form developer to provide the choices.

```xml
<elementLibrary>
  <LibraryElement id="libId70">
    <elementName><![CDATA[Subject]]></elementName>
    <displayName><![CDATA[Subject/Title/topic]]></displayName>
    <elementType>string</elementType>
    <elementBriefDescription><![CDATA[Topic of the Lesson Learned]]></elementBriefDescription>
    <detailedDescription><![CDATA[The Subject or Title reflects the primary topic of the Lesson Learned. A carefully written Subject should contain some of the Topic(s) that will help other users find this lesson in the future.]]></detailedDescription>
    <presentationStyle><![CDATA[mystyle.xsl]]></presentationStyle>
    <approved><![CDATA[true]]></approved>
  </LibraryElement>
  ...
</elementLibrary>
```

Figure 4  Example XML for storing element information

A type of special import is the Aggregate type. Choosing a type Aggregate lets the form developer group together a number of existing elements from the library to form a new element. Aggregate elements can contain other aggregate elements. For example, ‘Address’ can be an aggregate-type element consisting of elements Street, City, State/Province and Postal Code. For types Choices and Aggregate, the library stores additional information, i.e. the choices and the constituents respectively (not shown in the snippet in figure
4). The aggregate elements provide the advantage of creating pre-defined forms from the elements in the library. These can then be made available as recommended forms. If new formats for lesson submission are suggested, they can be easily created using the same form developer interface and made available for use. The underlying association of elements, aggregate elements and forms follows the Composite Pattern \[\] and is given in Figure 5.

![UML representation of elements](image)

After generation of the desired form, a lesson author can fill in his/her lesson and submit it. The submitted lesson is stored as an XML file in an XML repository, and an XML schema for the lesson file is dynamically generated.

It is possible to have a search mechanism that is linked to the library of elements (stored in an XML file). These elements are nothing but content holders in the lesson files. The search interface can provide users with the options for searching the supplied keywords in the content associated with some or all of the elements in the library. Newly added elements can be instantaneously reflected in the search interface, since the XML file containing the library of elements would be driving the interface.

The presentation of lessons can be also data-driven. One mechanism can be to come up with a set of presentation styles for whole lessons. For example, one style could be rendering all element names in bold, followed by their content in regular font. Lesson authors can choose one of the available styles for presentation of their lessons. Another mechanism can be to associate a presentation style with each element. We have provided a place holder for this information in the description of elements (see figure 4). This information can be provided by the form developer when creating an element. Thus, rendering of lessons can also be driven by the XML file containing the library of elements. Thus, new forms created using Aggregate type would no longer require any changes to be made to the overall system. Another advantage of this process is the gradual development of a “controlled vocabulary” (in the form of elements in the library) for the organization. It’s a natural next step to develop an ontology with these terms.

5. RESULTS AND CONTRIBUTIONS

The DLLS is still an ongoing work. We believe the main contribution of the work so far has been providing a proof of concept. The Dynamic Forms concept can effectively meet the two goals it was intended for:

- First, provide increased expressiveness and flexibility to lesson submitters while preserving the benefits of a highly-structured Lessons Learned System that guides and controls user input.
- Secondly, solving the problem of creating new lesson formats while still keeping the lessons submitted in older formats available without changing the overall system.

An added benefit is improved guidance for lesson search to scientists and engineers.

We intend to conduct a planned study of user interaction with dynamic forms to evaluate their usefulness. We believe, in presence of recommended forms (present as aggregate elements in the library), most users would use those. A small number of users would create their own forms and fewer still would add new elements to the library. The DLLS increases the burden on approving authorities, since now they have to approve elements apart from lessons. But the advantage gained in terms of better expression of lessons and improved search can make the added work worth the effort.

In the future, we intend to incorporate ontologies and thesauri, which would help users, better understand the semantics of the elements in the library and to guide search.

6. REFERENCES


[5] Overview of Object Role Modeling:
http://www.orm.net/


[7] XForms 1.0, November 2002, Available at: