Using Domain-Specific Modeling to Generate User Interfaces for Wizards

Enis Afgan, Jeff Gray, Purushotham Bangalore
University of Alabama at Birmingham
Department of Computer and Information Sciences
1300 University Boulevard
Campbell Hall #131
++ 1 (205) 934-2213
{afgane, gray, puri}@cis.uab.edu

ABSTRACT
The rising adoption and incorporation of computers into everyday life requires human-computer interaction methods to be efficient and easy to understand. Simultaneously, complexities of underlying computer systems are increasing, inherently requiring deeper understanding and a detailed level of human-computer interaction methods. Software wizards are one important example from the category of tools that simplify this interaction. Through a simple, domain-specific, and targeted set of guided questions, wizards allow complex tasks to be completed quickly and simply. Tasks accomplished by wizards range from simple information collection to complex system configuration. Because wizards are task-specific, their lifespan is short and thus must be easily and quickly adapted such that the cost associated with wizard maintenance is minimized. This paper outlines such wizard requirements and provides a metamodeling approach to wizard generation. A domain-specific modeling language is presented, which has been shown to be helpful in the generation of domain-specific wizards that are capable of adapting to changing requirements.

Categories and Subject Descriptors
D.2.13 {**Software Engineering**}: Reusable Software – *Domain engineering, reusable libraries, reuse models*

General Terms
Design, Reliability, Experimentation, Languages.

Keywords
Metamodeling, wizards, automated wizard generation, user interfaces

1. INTRODUCTION
Many advances in technology emerge from mechanisms that hide underlying accidental complexities by introducing additional layers of abstraction. The renewed interest in domain-specific languages is an example of how higher levels of abstraction can assist end-users in describing concerns from the problem space of a particular domain, as opposed to adopting notations of a specific solution space (*e.g.,* use of a general-purpose programming language or middleware). Software wizards [1] are an additional technique that assist in simplifying computer usage and configuration. By guiding configuration and customization through a set of targeted questions, a wizard can assist in resolving many activities that previously involved lower level knowledge of the inner workings of a specific application. For example, software installation and computer diagnostic tools use wizards to obtain information from a user that is needed to perform configuration and analysis tasks. As the trend toward raising levels of abstraction continues, wizards offer a viable alternative to assist end-users in describing more complicated tasks that refer to their domain expertise. With supporting tools, intuitive interfaces and guided suggestions provided by wizards can accomplish many tasks that minimize the required expertise of specific technical spaces. A challenge emerges, however, with respect to how the actual wizards are designed and created. This paper describes a domain-specific modeling language (DSML) that assists in generating wizards.

Because wizards are domain and problem specific, they are often transient and temporary in nature. New versions and various compositions of wizards need to be created, each having a possibly short life span. As such, allocating significant effort to wizard creation is poor use of one's time and should be minimized. We realized that a significant improvement in wizard composition could be offered by providing a modeling approach that could be used by a domain expert to design targeted wizards. A domain user can use the generated wizard to create necessary artifacts based on the parameters supported through the wizard questions. By investigating this technique, two categories of users and generators emerged: (1) the domain expert who uses a DSML for specifying a wizard in his/her own domain; this user uses the DSML to concentrate on collecting desired information, (2) the domain user who uses the wizard that was designed by the expert; this user provides specific and targeted information.

The idea of wizard composition and guided information collection contributes to the trend of improved abstraction mechanisms for specifying application information. By applying model-driven engineering (MDE) [2] to wizard composition and generation, improvements to wizard development are introduced by eliminating complexity of hand-coding all the complex links between pages. Furthermore, different model compilers can be associated with the wizard DSML to generate wizards in many different formats (*e.g.*, HTML, Java), each of which can store the obtained data in different formats (*e.g.*, text, XML, VoiceXML).

We have devised and developed a DSML that enables generation of user interfaces associated with wizards. A domain selected as an example and used throughout this paper comes from the area of grid computing [3]. The Application Specification Language (ASL) [4] is an XML-based language allowing an application developer to describe functionality, installation, and invocation properties of their application that persist throughout the grid environment. We selected ASL as an example because composition of an ASL specification can be a time-consuming and error-prone task for the application developer (*i.e.,* wizard user). The use of a wizard to create an ASL document consists of constructing representations of the necessary language elements describing an application. An application developer may answer the wizard's questions with parameters that describe the corresponding application feature. We have found that a wizard helps to remove many of the errors in formatting an ASL document to ensure that the document was created correctly (*i.e.*, the application developer would be alleviated of a lot of typing and checking the correctness of XML tags).

The roles of each user and the tools that they use are highlighted in Figure 1. The DSML generates the corresponding wizard (including page formatting and composition), which is then used by the application developer to provide descriptive information about the application through a targeted set of questions. After completing the wizard, an XML document is created corresponding to the data that was collected (*i.e.,* correctly formatted, ASL schema conforming document).

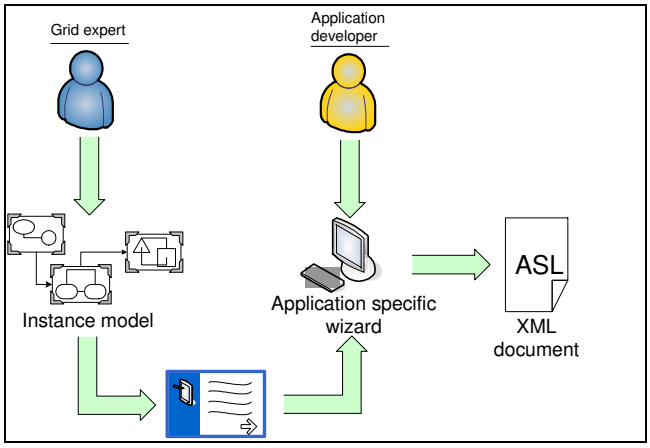

Figure 1. Two levels of abstraction accomplished by using a DSML to generate a higher level wizard.

2. WIZARD CLASSIFICATION

Using MDE to compose wizards requires the design of a metamodel capable of representing necessary entities within the wizard domain. Thus, determining the purpose of a metamodel depends on the type of wizards that will ultimately be created. In order to distinguish general types of metamodels to be created, we separate wizards into two broad categories: *plain* and *guided*.

Plain wizards correspond to simple page sequencing with appropriate fields incorporated into each page. The categorizing components of this type of metamodel are needed to enable and handle connectivity between different pages, sequencing of the pages, data passing and storage across the pages, as well as page design (including proper page formatting). There is a single path of execution built into this type of wizard at the time of model creation. After the model is created, the generator is in charge of converting the model into underlying code (*e.g.*, Java, HTML). Even though this type of wizard may seem somewhat trivial, the task that must be handled by the wizard generator at this stage is two-fold: a) create the user interface under given constraints; and b) simultaneously and automatically implement the method of capturing user data where it is output in the format (*e.g.*, text, XML, VoiceXML) specified by the user. Depending on the underlying technology used, the correct method of data passing must be applied (*e.g.*, if a wizard is created in HTML use CGI or servlets). This step of wizard generation must be implemented in the generator at a very generic level because the end-user denotes their intention by connecting two individual pages, but does not provide any additional parameters. The result of the modeling task is the specification of a software configuration wizard where the user specifies the necessary information to link appropriate libraries and the location of needed services.

Guided wizards extend the concept of page sequencing and are more closely related to expert systems [5]. Rather than having a predefined path set at the time of creation, guided wizards must have generic code incorporated into them so that user choices determine the next page of the wizard to be displayed. Incorporating these ideas into a metamodel requires much more care to be taken and imposes much higher requirements on the code generators. There are two major considerations at the metamodel level when dealing with a guided wizard. The first consideration is the requirement to create connections not only between entire pages of the wizard, but also to offer the user a set of predefined options. A user of the wizard modeling language must be able to make connections between those individual options and between subsequent wizard pages. The second consideration deals with page scoping. Because different paths in the course of wizard execution may take the wizard user to a page with equivalent information, there may be redundant pages floating around the instance model resulting in repetitive work done by the designer, eventually leading to more difficult page management and updating. Depending on the metamodel and corresponding data, this condition may be unavoidable, but by proper scoping and introduction of the hierarchical structuring of the collected output such issues can be reduced.

When using a generator for a wizard model and transforming it into the wizard implementation (*e.g.,* HTML or source code), there is the additional requirement to manage control flow elements for individual user choices. This logic must be customized to the particular wizard and be completely transparent to the end-user. Beyond making the necessary connections at the page level, the generator must be capable of composing necessary code, including page scoping which introduces new challenges for the generator developer. Figure 2 provides a graphical representation of these considerations, where the workflow of a guided wizard can be seen. The user initially provides some data, which leads to a subsequent page that may branch off into several possible pages. Selection of the subsequent page is determined on user input at run-time. Each page may branch into multiple subsequent pages, some of which can be equivalent in context, even though the paths may differ. At each step of the wizard, control flow (CF) code logic must be provided by the wizard generator along with the code for data storage that incrementally constructs the resulting document (*e.g.*, Data).

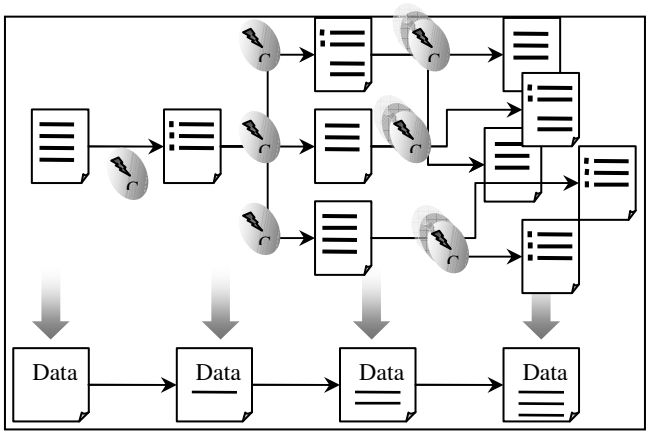

Figure 2. Wizard flow showing multiplicity of generated wizard pages and requirement to handle control flow (CF) at each junction as the Data document is being generated.

3. A PROTOTYPE METAMODEL

To investigate the benefits of model-driven generation of wizards, we developed a metamodel capable of representing wizard components found in ASL. A model compiler was also developed to generate the corresponding HTML code to represent the wizard. As a supporting tool, we used the Generic Modeling Environment (GME) [6], which is a metamodeling tool that can be used to build DSMLs. Our metamodel describes a modeling language that allows grid experts to specify the following elements of a wizard: compose pages with corresponding elements found in ASL, connect those pages into a meaningful flow, and generate matching HTML code that can be incorporated into a grid web portal interface.

The modeling language mirrors the structure of ASL. Because ASL is a hierarchically structured language, which is compartmentalized so that separate sections of a document are logically related, the metamodel conforms to the desired scoping rules of individual pages. By providing hierarchical components within the metamodel based on segregation of individual sections of ASL, the metamodel supports logical and meaningful structure of wizard generation to the grid expert. These sectionally-structured components provide page scoping, which allow the user to logically separate individual pages into subsections. This assists in keeping the number of pages to a manageable level. Because pages at separate sections of a wizard are generally dissimilar, metamodel segregation also minimizes the requirement for redundant page composition. At the current stage of development, the metamodel is capable of representing individual page components, compose those into a meaningful format and make them part of a larger section. Additionally, connections between entire pages corresponding to ASL sections can be established.

Figure 4 shows the metamodel with numbered elements, which are referenced in the rest of this section. In the metamodel, there are three major sections: section model, connection elements and individual page components. The section model (number 6) is the starting element of the entire metamodel setting the connectivity rules and encompassing all the other components available in the metamodel. The connectivity elements (numbers 1-5) establish different types of connections (*i.e.*, whether components belong to a single page or establish connections between pages), and the remaining elements (numbers 7-15) correspond to wizard page components. All of the elements except number 8 directly correspond to page components such as text box or a drop down menu. In the case of composite elements (*e.g.*, drop down menu, radio button group), lower level elements are provided to allow user creation of individual user options. Number 8 is a 'help element' allowing each element to be associated with context-sensitive help (as supplied by the metamodel user).

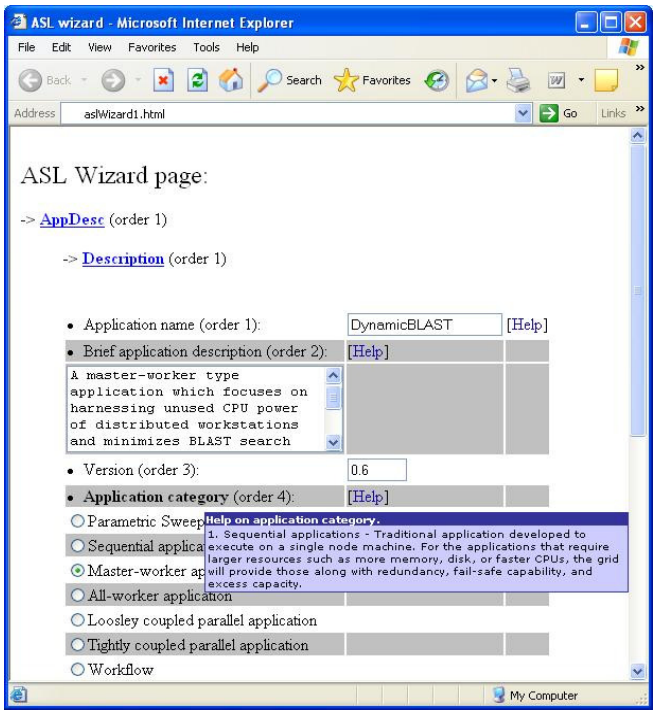

Figure 3. Sample page of generated wizard to collect application information as defined in ASL.

A model compiler was developed for this metamodel to generate the wizard in HTML and accommodate for proper page formatting, sequencing and transitioning. The development of the model compiler to support all available features of the corresponding metamodel presented a significant challenge. The current implementation of the model compiler is limited in functionality to support generation of HTML pages realizing the desired page formatting. Page arrangement and page connectivity is still unavailable. The major challenge arose from the need to automatically establish connectivity protocols based on different types of connected objects within a model. A sample page of the generated wizard in HTML is provided in Figure 3. This figure shows the initial page of the wizard, corresponding to the general information collection page of ASL. The figure also shows the metamodel supplied outline of the interactive help functionality.

4. CONCLUSIONS AND FUTURE WORK

This paper is a report of work in progress that started with a specific goal of producing wizards to assist in grid document and application configuration. During the course of our investigation, we realized that our modeling language was broader in scope and applicable to multiple domains. This paper provides a motivation of the requirements and challenges for specifying the composition

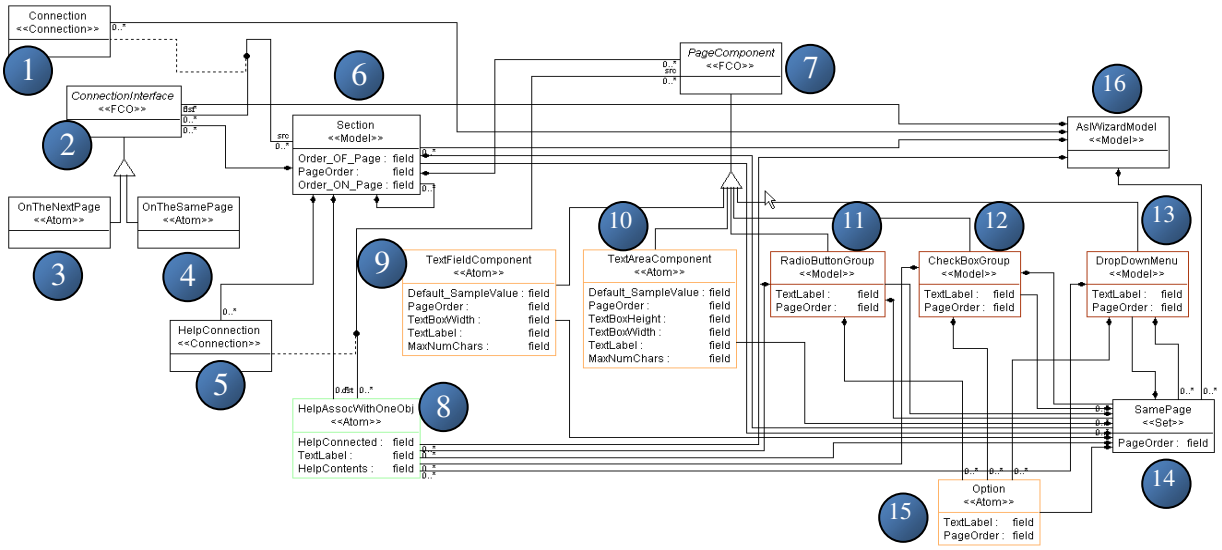

Figure 4. Metamodel for wizard generation. This particular instance is for creating ASL documents.

of generic wizards through the use of DSMLs. A contribution of this work is the two levels of indirection that have to be handled through the metamodeling environment. This means that the compiler must seamlessly produce not only the code for the wizard composition, but it also must incorporate code into the wizard that will store the data provided to the wizard at run-time in the appropriate format. The data storage component is not explicitly defined in the instance model and thus must exist in the generator. Provisioning of such functionality requires broad generality in the generator that must be capable of automatically invoking (i.e., as implied by current environment and/or user input) appropriate code. Development of such generalized code requires much interdependence between various code modules as well as complex code generation two levels deep (i.e., code generated by the model compiler is the wizard code). This generated code must subsequently be capable of responding to wizard requirements and guide the wizard actions as well as generate the final output.

A DSML was designed to support the approach. In the development process, issues arose with the implementation of the compiler code dealing with the support for the two levels of indirection. Because both of the mentioned levels must simultaneously be supported in the compiler code accommodating for any possible combination of elements found in the metamodel, the compiler encountered several challenges. This was due to the fact that at each step of the wizard and at each level of indirection very specific code needed to be created automatically. Portions of generator code were often found to be equivalent and dependent on dispersed snippets of code inviting generator code components to be indirectly connected. Use of standard object-oriented programming techniques resulted in inconsistent functionality that was hard to manage and maintain. More specifically, this was referring to compiler code that was used to generated page connections and organize page layout. Because each component was slightly different depending on the context, compiler code needed to be adjusted accordingly rather than simply being able to reuse it transparently. A need to modularize code into more manageable components, each capturing the desired functionality, became apparent. Thus, we are considering the use of techniques such as the Aspect-Oriented Programming [7] to solve such issues. Eventually, with the improvements in generation of our metamodel compiler, new fields for tool applicability can be realized, such as automated job submission interface generation for grid applications, as well as simultaneous output of multiple formats of wizard-collected data requiring minimal user intervention.

5. REFERENCES

[1] D. Batory, G. Chen, E. Robertson, and T. Wang, "Web-Advertised Generators and Design Wizards," *International Conference on Software Reuse (ICSR)*, Victoria, Canada, 1998.

[2] D. Schmidt, "Model-Driven Engineering," *IEEE Computer*, vol. 39, pp. 25-32, 2006.

[3] *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 1998.

[4] E. Afgan and P. Bangalore, "Application Specification Language (ASL) – A Language for Describing Applications in Grid Computing," *4th International Conference on Grid Services Engineering and Management (GSEM)*, Leipzig, Germany, 2007.

[5] J. Durkin, *Expert Systems: Design and Development*, Macmillan, 1998.

[6] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, and S. Neema, "Developing Applications using Model-Driven Design Environments," *IEEE Computer*, vol. 39, pp. 33-40, 2006.

[7] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin, "Aspect-Oriented Programming," *European Conference on Object-Oriented Programming (ECOOP)*, Jyväskylä, Finland, 1997.