SMARTFORM: A Web-based Feature Configuration Tool

Wonseok Chae
Toyota Technological Institute at Chicago
wchae@tti-c.org

Timothy L. Hinrichs
University of Chicago
tlh@uchicago.edu

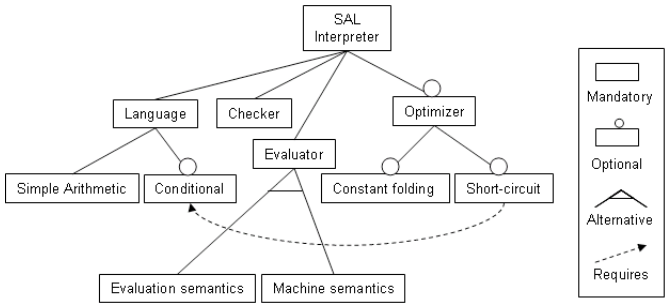

Fig. 1. Feature model for the SAL interpreter (from [4]).

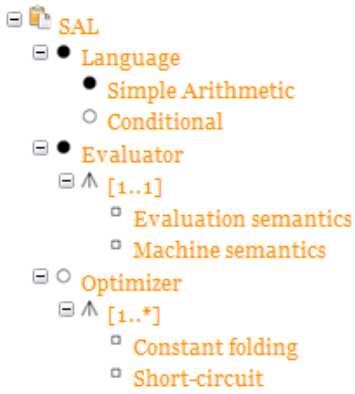

Fig. 2. SPLOT (Software Product Lines Online Tools) provides a feature modeling editor similar to that of FeaturePlugin [14]. Cross-tree constrains are separately specified in a conjunctive normal form. In this example, $\neg$ (Short $-$ circuit) $\vee$ Conditional specifies the constraint that the selection of Short $-$ circuit requires the selection of Conditional.

***Abstract*—In feature-oriented software development, features distinguish different applications and a feature model abstractly represents the set of all possible applications for a given domain. Selecting a set of features that obey the feature model is the first step toward the synthesis of an application. In this paper, we present SMARTFORM, a web-based feature configuration tool that facilitates the process of feature selection. Its web-based feature selection user interface utilizes a web form generation tool PLATO to perform real-time validation.**

***Keywords*-feature modeling, variability, configuration.**

I. INTRODUCTION

Feature-oriented software development (FOSD) is an emerging paradigm for designing an entire domain of applications from a set of reusable software components. Each application in the domain is composed from a different subset of those components, and features are used to distinguish each application in the domain. Feature models represent all the permitted combinations of features, and selecting features that obey the feature model is the first step toward the synthesis of an application [10].

In previous work, we demonstrated that FOSD is an effective way to build a family of programs and showed that the process of combining software components given a feature selection can be automated using advanced implementation techniques [3], [4]. Those techniques assume that the given feature selection is valid, i.e., that it obeys the feature model, but when feature models are large or complex, this assumption can be difficult to satisfy.

One approach that helps significantly is making users understand the feature model before they begin feature selection. The traditional representation of a feature model (a feature diagram, e.g. Figure 1), is well-recognized to be inadequate for large examples, though few alternative representations have been introduced. One notable exception is the FeaturePlugin tree representation, e.g., Figure 2, which allows for efficient entry and editing of large feature diagrams [1]. Unfortunately, for certain product domains, feature models can be so large that even the tree representation has been found inadequate. Additionally, even if the feature model is well-understood, it can be time-consuming to manually validate a given feature selection, especially when the feature model includes additional constraints (e.g., the selection of one feature precludes or requires the selection of others). In response some researchers have investigated techniques for reducing the complexity of feature models (e.g., this year's workshop on Scalable Modeling Techniques on Software Product Lines [13]), while others have focused on tools and methodologies that relieve both users and application developers from performing the task.

Automated tools have the potential to be of great benefit to users attempting to make valid feature selections. One branch of research is devoted to techniques for automated feature selection [15], though there is evidence that users would prefer to work interactively with feature-solicitation systems so they retain control over the feature selection process while leveraging the power of automation [9]. Another branch of

research aims at automatically validating a feature selection against a feature model. For example, recent work shows how off-the-shelf tools (e.g., the SVM systems, SAT solvers or CSP solvers) can be used to validate a feature selection by transforming the feature model to an appropriate decision problem [12], [2], [15], [8], [6]. This approach has the drawback that off-the-shelf tools are often difficult to deploy in settings for which they were not designed. For example, consider building a web-based collaborative feature selection tool where many users can simultaneously make (partial) feature selections and have those selections validated against a feature model automatically. To use an off-the-shelf SAT solver for feature validation, the SAT solver would need to be installed and run on the server. Thus under heavy usage, the server might be required to solve tens or even hundreds of SAT problems simultaneously, and since solving a single SAT problem is NP-hard, such an approach to collaborative feature selection would not scale.

In this paper, we explore the groundwork for building a web-based feature configuration tool that facilitates the process of feature selection. We render a feature digram in tabular form and employ checkboxes and radio buttons to allow multiple and singleton selections, respectively. This basic interface appeals to users not familiar with feature diagrams but tasked with making feature selections. Implementationally, our application offloads the work of feature selection validation to the clients, ensuring scalability. Moreover, feature selection validation is performed each time a user changes any portion of her selection, thereby allowing a user to understand at each choice point how that choice affects the overall validity of her selection. Our tool, SMARTFORM, builds upon a generic web form generation tool, PLATO [7], which constructs forms that provide users visual feedback about errors and implied values as they enter data.

II. PLATO: A FORM GENERATION SYSTEM

PLATO is a tool for automatically generating web forms from logical descriptions of those forms. A web form designer declares the desired properties for each form field and specifies the relationships between fields in logic. PLATO compiles the designer's description into HTML and Javascript, complete with error-checking and value-propagation code. To understand the underlying mechanisms, it is instructive to first look at an example.

TABLE I
THE DECLARATION OF FORM FIELDS.

Name	Style	Type	Description
C	checkbox	boolean	Conditional
E	radio	enum {Evaluation, Machine}	Evaluator
O	checkbox	boolean	Optimizer
CF	checkbox	boolean	Constant folding
SC	checkbox	boolean	Short-circuit

Consider building a web form for the SAL interpreter feature model. First, we decide how to represent the features

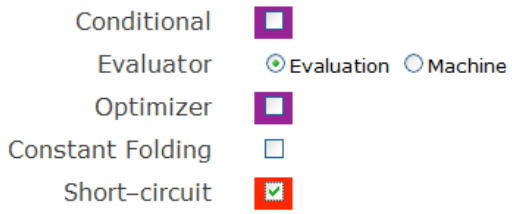

Fig. 3. An example output of PLATO, which converts form descriptions into HTML and Javascript. An erroneous feature selection and its causes are displayed in different colors.

in the model as fields on the web form. Checkboxes are useful for optional features and radio buttons are good for alternative features. Table I includes five form fields (C, E, O, CF, SC) representing six features (Conditional, Evaluation, Machine, Optimizer, Constantfolding, Short − circuit).

Second, we specify all the constraints that the feature model imposes on the web form fields. In PLATO, constraints are written in a well-behaved fragment of first-order logic (FOL). Our running example requires three constraints:

- The selection of Optimizer requires the selection of either Constant folding or Short − circuit. At the same time, unless Optimizer is selected, neither of its children can be selected. These constraints are specified as follows:

```
(<=> O (or CF SC))
```

  Here, O, CF and SC refer to the name of fields we defined in Table I. *or* and <=> are logical operators, which give the logic above the following meaning. "O is true if and only if either CF or SC or both are true."
- The selection of Short − circuit requires the selection of Conditional.

```
(=> SC C)
```

- Either Evaluation semantics or Machine semantics must be selected. This constraint is enforced directly by the radio button but could be included explicitly.

PLATO compiles the two items above (field descriptions and logical constraints) into HTML and Javascript that implements error-detection and value-propagation code. Figure 3 shows the result that PLATO generates. It also illustrates an invalid feature selection (selecting Short − circuit without selecting Conditional and Optimizer) and the visual cues PLATO uses to identify errors and their causes.

III. SMARTFORM

The SMARTFORM tool is structured as shown in Figure 4. It takes a feature model written in a simple XML feature model format (SXFM) and produces a web form for feature selection that performs validation each time the user makes a selection. Internally, SMARTFORM transforms the feature model into a logical web form description and passes the result to PLATO. It also generates layout information, which when combined with PLATO's output produces the final web

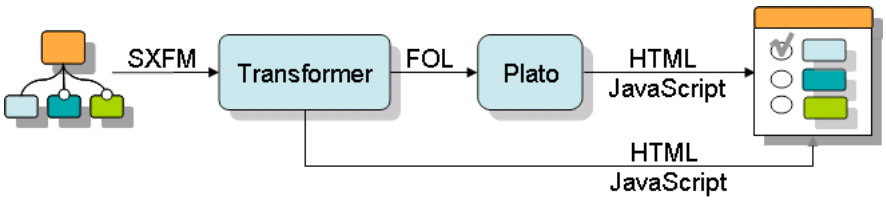

Fig. 4. The overall system for the SMARTFORM generator.

form for feature selection. In this section, we discuss each component of SMARTFORM.

A. Simple XML Feature Model

The Generative Software Development Lab at the University of Waterloo launched the Software Product Lines Online Tools (SPLOT) to put software product line research into practice [14]. They also introduced a simple XML feature model format (SXFM) that makes it easy to define feature models using a simple text editor and illustrated their format with a library of about 30 models. So that we can utilize these models and their advanced feature model editor, we use the SXFM format as our primary form of input.

Figure 5 shows how our running example can be specified in SXFM. It consists of three sub-elements: meta, feature_tree and constraints. The meta section is self-explanatory. The feature_tree section corresponds to a feature diagram where the levels in the diagram are represented by tabulations instead of tags to keep the size of XML files small. The constraints section defines constraints not represented by the feature diagram directly in conjunctive normal form. In this example, the constraint that "the selection of Short − circuit requires the selection of Coditional" is represented by "˜_r_3_9_11 or _r_1_5" simply meaning that " _r_3_9_11 implies _r_1_5".

```
<feature_model name="SAL">
<meta>
<data name="description">Simple Arithmetic Language</data>
</meta>
<feature_tree>
:r SAL(_r)
        :m Language(_r_1)
                :m Simple Arithmetic(_r_1_4)
                :o Conditional(_r_1_5)
        :m Evaluator(_r_2)
                :g (_r_2_6) [1,1]
                        : Evaluation semantics(_r_2_6_7)
                        : Machine semantics(_r_2_6_8)
        :o Optimizer(_r_3)
                :g (_r_3_9) [1,*]
                        : Constant folding(_r_3_9_10)
                        : Short-circuit(_r_3_9_11)
</feature_tree>
<constraints>
constraint_2: ~_r_3_9_11 or _r_1_5
</constraints>
</feature_model>
```

Fig. 5. This model is created online using SPLOT's Feature Model Editor.

B. Transformer engine

The Transformer engine parses a feature model in SXFM format into (i) HTML layout templates and (ii) a logical web form description for PLATO. The first item, conversion of the feature diagram to a tabular structure is straightforward. The only subtlety that arises is when the children of some parent are constrained as optional selections or as alternative selections. We treat such children as checkboxes and radio buttons, respectively.

As illustrated in Section II, PLATO requires a logical description of the form fields and the constraints on those fields. The web form description consists of three sections: widgets, types, and constraints. See Figure 6. A widget declaration defines the display properties for each form field. PLATO supports various styles (e.g., checkboxes and radio buttons), but since SMARTFORM generates its own layout information, this section can be ignored. The type section requires information about the possible values for each field: string, boolean, or an enumerated list, information that is easily extracted from the SXFM feature model. The constraints section is the most interesting section to construct. In addition to the "constraints" element of the SXFM format, SMARTFORM includes structural constraints implicit in the feature model such as parent-child relationships. These structural constraints must be made explicit for PLATO since it is a general-purpose tool and includes no information about the semantics of feature-models.

In our example, Optimizer is the parent of Constant folding and Short − circuit, which causes SMARTFORM to include the constraint requiring at least one of Constant folding or Short − circuit to be selected if Optimizer is selected.

Once the logical web form description has been assembled, the transformer performs two tasks: it computes the HTML layout of radio buttons and checkboxes and sends the web form description to PLATO through a SOAP request.

C. SMARTFORM

SMARTFORM embodies the combination of SXFM, PLATO, and the Transformer engine. The results of the transformer engine and PLATO are combined to construct a web form for feature selection where feature validation is performed in real-time by the web browser. For instance, when a user selects Short − circuit without selecting Conditional and Optimizer, the form immediately highlights the violations, pointing to the

```
(WIDGET :NAME _r_1_5     :INIT "" :DESC "Conditional"          :STYLE CHECKBOX ) (TYPE _r_1_5     BOOLEAN)
(WIDGET :NAME _r_2_6_7   :INIT "" :DESC "Evaluation semantics" :STYLE CHECKBOX ) (TYPE _r_2_6_7   BOOLEAN)
(WIDGET :NAME _r_2_6_8   :INIT "" :DESC "Machine semantics"    :STYLE CHECKBOX ) (TYPE _r_2_6_8   BOOLEAN)
(WIDGET :NAME _r_3       :INIT "" :DESC "Optimizer"            :STYLE CHECKBOX ) (TYPE _r_3       BOOLEAN)
(WIDGET :NAME _r_3_9_10  :INIT "" :DESC "Constant folding"     :STYLE CHECKBOX ) (TYPE _r_3_9_10  BOOLEAN)
(WIDGET :NAME _r_3_9_11  :INIT "" :DESC "Short-circuit"        :STYLE CHECKBOX ) (TYPE _r_3_9_11  BOOLEAN)
(CONSTRAINTS '((=> _r_3_9_10 _r_3) (=> _r_3_9_11 _r_3) (OR (_r_1_5) (NOT _r_3_9_11))))
```

Fig. 6. The transformer engine converts a feature model to a logical web form description for PLATO. While the description shown above is Lisp-like, PLATO is scheduled to support a purely XML input format in the near future.

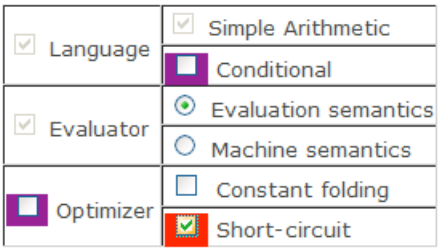

Fig. 7. The SMARTFORM highlights two errors: 1) the Short − circuit feature requires the Conditional feature; 2) the Short − circuit feature cannot be solely selected without selection of its parent feature.

errors as illustrated in Figure 7.

IV. CONCLUSION

In this paper, we presented a web-based feature configuration tool that facilitates the process of feature selection. The user-interface produced by our tool includes feature validation implemented entirely in the browser that executes each time a user changes their selection, giving users real-time feedback.

In the future, we plan to extend SMARTFORM to enable tens, hundreds, or even thousands of users to collaboratively make a single feature selection. We hope to support a broad range of well-understood methodologies for collaborative feature selection, e.g. multiple views or multi-staging [11], [5]. Two of SMARTFORM's features were put in place to make extensions that support collaboration straightforward.

First, SMARTFORM was designed as a web application. By deploying on the web, users all over the world can participate in the feature selection process almost immediately because they only need access to a standard web browser and Internet connection. As an added benefit, because they are using web browsers, users are already familiar with basic operations and will become comfortable far more quickly than they would with custom software. Additionally, we as implementors can utilize a plethora of tools and methodologies designed to simplify the maintenance and construction of massively parallel applications.

Second, SMARTFORM produces web forms that allow users to make feature selections that violate the feature model (but inform them of the mistake). Such an interface aids collaboration because it is capable of representing the joint feature selection of numerous users. When collaboratively making a feature selection, a user might want to start from scratch, or she might want start from the current joint progress of all her colleagues. Ideally, she would use the same interface in either case, but that requires the interface to represent disagreements among collaborators, which are violations of the feature model. Thus, by allowing an individual user to violate the feature model (at least temporarily), our tool paves the way for collaboration.

REFERENCES

[1] Michal Antkiewicz and Krzysztof Czarnecki. FeaturePlugin: feature modeling plug-in for eclipse. In *OOPSLA workshop on eclipse technolgy exchange*, New York, NY, USA, 2004.

[2] Don Batory. Feature models, grammars, and propositional formulas. In *Proceedings of the International Software Product Line Conference*, pages 7–20, 2005.

[3] Wonseok Chae and Matthias Blume. Building a family of compilers. In *Proceedings of the 12th International Software Product Line Conference*, 2008.

[4] Wonseok Chae and Matthias Blume. Language support for feature-oriented product line engineering. In *Proceedings of the First International Workshop on Feature-Oriented Software Development*, pages 3–10, 2009.

[5] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[6] Abdelrahman Osman Elfaki, Somnuk Phon-Amnuaisuk, and Chin Kuan Ho. Knowledge based method to validate feature models. In *Proceedings of the International Software Product Line Conference*, 2008.

[7] Timothy Hinrichs, Jui-Yi Kao, and Michael Genesereth. Automatic web form construction via paraconsistent compilation to relational databases. Technical report, University of Chicago, 2010.

[8] Mikolas Janota and Joseph Kiniry. Reasoning about feature models in higher-order logic. In *Proceedings of the 11th International Software Product Line Conference*, pages 13–22, 2007.

[9] Mikoláš Janota, Goetz Botterweck, Radu Grigore, and Joao Marques-Silva. How to complete an interactive configuration process? In *Proceeding of 36th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, 2010.

[10] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. FORM: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, 1998.

[11] Kwanwoo Lee, Kyo Chul Kang, and Jaejoon Lee. Concepts and guidelines of feature modeling for product line software engineering. In *Proceedings of the 7th International Conference on Software Reuse*, pages 62–77, 2002.

[12] Mike Mannion. Using first-order logic for product line model validation. In *Proceedings of the Second International Conference on Software Product Lines*, pages 176–187, 2002.

[13] Workshop on Scalable Modeling Techniques for Software Product Lines. http://kishi-www.jaist.ac.jp/SCALE2009/, 2009.

[14] Software Product Lines Online Tools. http://www.splot-research.org/, 2009.

[15] Jules White, Douglas C. Schmidt, Egon Wuchner, and Andrey Nechypurenko. Automating product-line variant selection for mobile devices. In *Proceedings of the International Software Product Line Conference*, pages 129–140, 2007.