Feature Diagrams as Package Dependencies*

Roberto Di Cosmo and Stefano Zacchiroli

Université Paris Diderot, PPS, UMR 7126, Paris, France
roberto@dicosmo.org,zack@pps.jussieu.fr

Abstract. FOSS (Free and Open Source Software) distributions use dependencies and package managers to maintain huge collections of packages and their installations; recent research have led to efficient and complete configuration tools and techniques, based on state of the art solvers, that are being adopted in industry. We show how to encode a significant subset of Free Feature Diagrams as interdependent packages, enabling to reuse package tools and research results into software product lines.

Keywords: software product lines, open source, package, component, feature diagram, dependencies

1 Introduction

Feature models [11] are essential devices to reason about software product lines (SPLs). As features and their interdependencies get more complex, managing models quickly turns into a non-trivial task for humans. Researchers have worked to improve the situation by establishing connections between Feature Diagrams (FD) and notions like grammars [6], propositional logics [2], and constraint programming [3], paving the way to use automatic tools like SAT solvers [9,12] and proof assistants [10] for SPL con guration.

Meanwhile FOSS distributions like Debian, Red Hat, and Suse have used for the past 15 years packages and dependencies to maintain some among the largest software collections known: package managers are used daily to maintain millions of installations built by selecting components from repositories of tens of thousands packages. Recent research e orts [1,4,13,16] have led to the development of e cient and complete con guration and maintenance tools, as well as metrics for FOSS distributions, based on state of the art solvers, which are rapidly being adopted in industry.

We show how a signi cant subset of Free Feature Diagrams can be compactly encoded as interdependent packages, opening the way to massive reuse in SPLs of research results and tools coming from FOSS research. In particular, package management tools are able to scale up to tens of thousands components and hundreds of thousands dependencies, and cope well with component evolution, which is routine in FOSS. Both aspects may be of great interest for the SPL community.

* This work is partially supported by the European Community FP7, MANCOOSI project, grant agreement n. 214898.

2 Package dependencies for distribution maintenance

In FOSS distributions a *package* is a bundle that ships a (software) component, the data needed to configure it, and metadata which describe its attributes and expectations on the deployment environment [7]. For simplicity, we focus on packages as used in the Debian distribution, but the discussion applies almost unchanged to other popular package formats like RPM (see [13] for details).

Here is a sample metadata excerpt from the `firefox` package:

```
Package: firefox
Version: 1.5.0.1-2 ...
Depends: fontconfig, psmisc, libatk1.0-0 (>= 1.9.0), libc6 (>= 2.3.5-1) ...
Suggests: xprint, firefox-gnome-support (= 1.5.0.1-2), latex-xft-fonts
Conflicts: mozilla-firefox (<< 1.5-1)
Replaces: mozilla-firefox
Provides: www-browser, ...
```

Every package has a version that is used to give a temporal order to the packaged component release. The kinds of relationships that can be expressed in the metadata of a package `p` are numerous, but the most important are:

- `Depends`: a list of package disjunctions $p_1 \mid \ldots \mid p_n, \ \ldots, q_1 \mid \ldots \mid q_m$, where each atom can carry a version predicate (e.g. $\geq$ 1.9.0). For the owner package to be installable, at least one package in each disjunction must be installed.
- `Conflicts`: a list of package predicates $p_1, p_2, \ldots p_n$. For the owner package to be installable, none of the p_i must be installed. *Self conflicts are ignored.*
- `Recommends` similar to `Depends`, it indicates an optional dependency; it might be advisable to satisfy it, but it is not needed to obtain a working system.

A *repository* R is a set of packages; a subset $I \subseteq R$ of it is said to be a *healthy installation* if all dependencies of packages in I are satisfied, and none of the conflicts is. Precise formal meanings to all these notions have been given elsewhere (see [8,13] and the Mancoosi project http://www.mancoosi.org). Tools are available in the distribution world to choose healthy installations according to user requests [15] and to perform sophisticated repository analysis [1,5].

3 Encoding feature diagrams as package dependencies

We show how a core subset of Feature Diagrams (FD) can be compactly encoded as packages. Due to the differences among FD formalisms, we provide the encoding for a significant subset of Free Feature Diagrams (FFD) [14] that captures many known formalisms, and allows to claim that our encoding is of general interest.

FFD is a general framework that allows to capture different classes of FD by specifying a few parameters: the kind of graph GT (Tree or DAG); the node type NT (and, or, xor, or opt; the latter encoding explicit optionality within a node-based semantics [14]); the graphical constraint type GCT ($\Rightarrow$ for implication, and $\mid$ for mutual exclusion); and the textual constraint language TCL (usually including just implication and mutual exclusion, noted $n \ implies \ n'$ and $n \ mutex \ n'$, respectively).

Definition 1 (Free Feature Diagram). $d \in FFD(GT, NT, GCT, TCL) = (N, P, r, \lambda, DE, CE, \Phi)$ *where:*

- N *is a set of nodes*
- $P \subseteq N$ *is a set of primitive nodes*
- $r \in N$ *is the root node*
- $\lambda : N \to NT$ *labels each node with an operator from* NT
- $DE \subseteq N \times N$ *is the set of decomposition edges;* $(n, n') \in DE$ *is noted* $n \to n'$
- $CE \subseteq N \times GCT \times N$ *is the set of constraint edges*
- $\Phi \subseteq TCL$ *are the textual constraints*

A few well-formedness constraints are imposed: only r has no parent; the decomposition edges do not contain cycles; if GT is Tree, then DE forms a tree; nodes are labeled with operators of the appropriate arity.

Precise formal semantics of FFD is given in terms of valid models [14]:

Definition 2 (Valid model). *A* valid model *of a feature diagram* d *is* $M \subseteq N$ *such that: (a)* $r \in M$*, (b)* M *satisfies the operators attached to each node, as well as all the (c) graphical and (d) textual constraints, with the additional requirement that (e) if a node is in the model, then at least one of its parents (called the* justi cation*) is in the model too.*

We call FFD_{core} the fragment of FFD obtained by restricting the operators in NT to or, and, xor, opt, and the operators in GCT and TCL to $implies$ and $mutex$; this fragment is enough to cover several well known Feature Diagram formalisms (OFT, OFD, RFD, VBFD, GPFT and PFT in the classi cation given in [14]).

We will now show how to encode any $d \in FFD_{core}$ as a package repository $R(d)$, so that valid models correspond to healthy installations of a speci c package $p_r \in R(d)$.

Definition 3 (FFD_{core} encoding). *Let* $d \in FFD_{core}(GT, NT, GCT, TCL) = (N, P, r, \lambda, DE, CE, \Phi)$*, we define the package repository* $R(d)$ *as follows:*

- *the packages* P *are defined as* $\{(n, 1) | n \in N\}$*, so we have a package for each node in the diagram, with a unique version number,* 1
- $\forall$ *node* n *with sons* $n_1, \ldots, n_k$*, add dependencies as follows:*
  - *if* $\lambda(n) =$ *or, add* $n_1, \ldots, n_k$ *as disjunctive dependencies for* n
  - *if* $\lambda(n) =$ *and, add* $n_1, \ldots, n_k$ *as conjunctive dependencies for* n
  - *if* $\lambda(n) =$ *xor, add* $n_1, \ldots, n_k$ *as disjunctive dependencies of* n*, and add* $n_1, \ldots n_{i-1}, n_{i+1}, \ldots, n_k$ *as conflicts for* $\forall n_i$
- $\forall$ *node* n *with son* n' *and* $\lambda(n) =$ *opt, add* n' *as a recommend of* n
- $\forall$ *constraint* c *in* CE *or* Φ*, add the following dependencies:*
  - *if* c *is* $n \Rightarrow n'$ *or* n *implies* n'*, add* n' *as a conjunctive dependency of* n
  - *if* c *is* $n | n'$ *or* n *mutex* n'*, add* n' *to the conflicts of* n
- *if* $GT =$ *Tree,* $\forall n \to n' \in DE$*, add* n *as a conjunctive dependency for* n'
- *if* $GT =$ *DAG,* $\forall n \neq r$*, add a dependency on* $n_1 \mid \cdots \mid n_k$ *where* $n_1, \ldots, n_k$ *are all the parents of* n

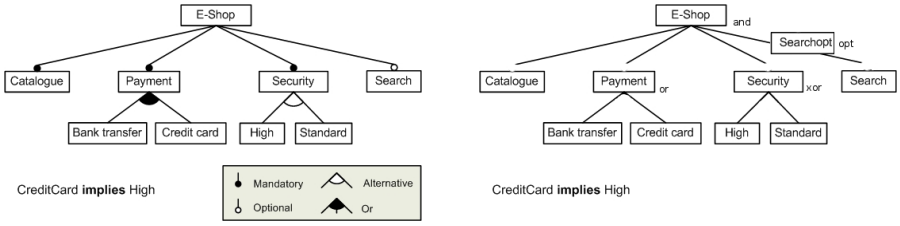

Fig. 1. Sample E-Shop feature model: as FD (on the left) and FFD (on the right).

An even more compact, linear-space encoding for justi cations and *xor* nodes can be given using *virtual packages* [7], exploiting the property that self-con icts are ignored; it has been omitted here due to space constraints.

Example 1. A feature model using an edge-based semantics for an e-shop is shown in Figure 1 as FD (on the left) and FFD (on the right). Its encoding as package repository is reported below, where we drop `Version: 1`.

```
Package: E-Shop
Depends: Catalogue, Payment, Security, SearchOpt

Package: Catalogue
Depends: E-Shop

Package: Payment
Depends: BankTransfer | CreditCard, E-Shop

Package: BankTransfer
Depends: Payment

Package: CreditCard
Depends: Payment, High

Package: Security
Depends: High | Standard, E-Shop

Package: High
Depends: Security
Conflicts: Standard

Package: Standard
Depends: Security
Conflicts: High

Package: SearchOpt
Depends: E-Shop
Recommends: Search

Package: Search
Depends: SearchOpt
```

Notice how all kinds of metadata are used: con icts encode *mutual exclusion*, recommends encode optional features, conjunctive depends encode *and* nodes and implications, disjunctive dependencies encode *or* nodes. It is now possible to establish the key property of the detailed encoding.

Theorem 1 (Soundness and completeness). *A subset $M \subseteq N$ of the nodes of a $d \in FFD_{core}$ is a valid model of d if and only if m is a healthy installation for the package repository encoding $R(d)$.*

Proof. The proof is by case analysis on the definition of a valid model, and the structure of the encoding. Details are omitted due to lack of space.

4 Conclusions and future work

We have established a direct mapping from a signi cant subset of Free Feature Diagrams to packages of FOSS distributions. This paves the way to reuse of theo-

retical results as well as tools coming from FOSS research. Package management tools scale to tens of thousands packages and hundreds of thousands dependencies, and cope with evolving components. For instance, the edos.debian.net site provides quality metrics for FOSS distributions comprising more than 20'000 packages and 400'000 dependencies, daily, since 2006. Also, model construction with respect to user-defined optimizations is implemented by several tools, and competitions like www.mancoosi.org/misc-2010 are improving their efficiency.

We plan to extend the current encoding to all FFD constructs such as cardinality constraints and to validate the proposed approach by providing a full toolchain that attacks SPL problems using existing package management technology.

We hope that our work will contribute to establish a connection between SPLs and package management, for the joint benefit of both communities.

References

1. Abate, P., Boender, J., Di Cosmo, R., Zacchiroli, S.: Strong dependencies between software components. In: ESEM 2009. pp. 89–99. IEEE (2009)
2. Batory, D.: Feature models, grammars, and propositional formulas. In: SPLC 2005. LNCS, vol. 3714, pp. 7–20. Springer (2005)
3. Benavides, D., Martín-Arroyo, P.T., Cortés, A.R.: Automated reasoning on feature models. In: CAiSE. LNCS, vol. 3520, pp. 491–503. Springer (2005)
4. Berre, D.L., Rapicault, P.: Dependency management for the Eclipse ecosystem. In: IWOCE'09. ACM (2009)
5. Boender, J., Di Cosmo, R., Vouillon, J., Durak, B., Mancinelli, F.: Improving the quality of GNU/Linux distributions. In: COMPSAC. pp. 1240–1246. IEEE (2008)
6. de Jonge, M., Visser, J.: Grammars as feature diagrams. In: ICSR7 Workshop on Generative Programming. pp. 23–24 (2002)
7. Di Cosmo, R., Trezentos, P., Zacchiroli, S.: Package upgrades in FOSS distributions: Details and challenges. In: HotSWup'08. ACM (2008)
8. EDOS project, WP2 team: Report on formal management of software dependencies. Deliverable Work Package 2, Deliverable 2 (2006)
9. Janota, M.: Do sat solvers make good configurators? In: SPLC 2008, Second Volume (Workshops). pp. 191–195 (2008)
10. Janota, M., Kiniry, J.: Reasoning about feature models in higher-order logic. In: SPLC. pp. 13–22. IEEE Computer Society (2007)
11. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study. Tech. rep., CMU (1990)
12. Le Berre, D., Parrain, A.: On SAT technologies for dependency management and beyond. In: ASPL'08 (2008)
13. Mancinelli, F., Boender, J., Di Cosmo, R., Vouillon, J., Durak, B., Leroy, X., Treinen, R.: Managing the complexity of large free and open source package-based software distributions. In: ASE 2006. pp. 199–208. IEEE (2006)
14. Schobbens, P.Y., Heymans, P., Trigaux, J.C.: Feature diagrams: A survey and a formal semantics. In: RE'06. pp. 136–145. IEEE (2006)
15. Treinen, R., Zacchiroli, S.: Solving package dependencies: from EDOS to Mancoosi. In: DebConf 8: proceedings of the 9th conference of the Debian project (2008)
16. Tucker, C., Shuffelton, D., Jhala, R., Lerner, S.: OPIUM: Optimal package install/uninstall manager. In: ICSE 2007. pp. 178–188 (2007)