Abstract

Many office and enterprise business applications are overloaded with features. As a result users struggle to find the functionality needed to support their tasks. Customization support for existing applications is often limited and hard to accomplish for end users. Software product lines provide support for customizing complex applications. However, they typically support vendors deriving customized products for customers from reusable components rather than supporting end users. In this paper we show how a decision-oriented software product line approach can support (1) software vendors deriving products for customers, (2) customers configuring products to the needs of specific user groups, and (3) end users personalizing a system to their needs. We describe tool support and illustrate the approach with a feasibility study.

1. Introduction and Motivation

End users of today’s office and enterprise business applications often struggle to understand the offered functionality. Typically, users only need a small fraction of the features and are overwhelmed by many unneeded capabilities. Best-in-class software manufacturers support tailoring systems to specific customers and also provide personalization support for end users. However, customization and personalization support is still quite limited for many applications and based on different techniques that are not well integrated. The situation is even more challenging for small- and medium-sized software vendors. For them, personalizing applications remains a challenging task as today’s development practices do not provide sufficient guidance or tool support to create highly customizable systems.

It has been demonstrated that software product lines provide a reasonable approach to customization of complex software systems [17, 26]. A software product line is “a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a predefined way” [4]. Software product line engineering (SPLE) deals with building, managing, and using software product lines and covers the areas of domain engineering and application engineering [17]: In domain engineering, the variability of a product line’s solution components is explicitly captured in variability models. In application engineering, customized products are derived from the product line by using these models and deploying the needed solution components. Product lines are beneficial if the costs of defining them are outweighed by the benefits of quick and correct derivation of customized products from the common core of shared, reusable assets [7]. So far, product lines have mainly been used by software producers to derive and deploy customized products for different customers. In this paper, we demonstrate that the use of product lines can be extended to provide personalization support for end users.

In our collaboration with several industrial and academic partners from the domains of industrial automation [21], enterprise resource planning [28], and service-oriented systems [5] end users have requested easy-to-use capabilities allowing them to personalize an application to their needs. In a product line approach variability models describe the possible choices for customization. Making these models accessible to end users requires role- and task-specific views on the offered variability. Furthermore, from a technical perspective, run-time adaptation mechanisms are required as end users should not have to worry about installation and system restarts after customization [28].

The remainder of this paper is structured as follows. In Section 2, we define three levels of product customization and present our approach and tool support for these levels. In Section 3, we present a feasi-
bility study of applying our approach and tools in the enterprise resource planning domain for our industrial partner BMD. In Section 4 we discuss related work. Section 5 rounds out the paper with conclusions and an outlook on future work.

2. Approach

Figure 1 depicts three levels of product customization and involved stakeholders. At each level, more variability is resolved as shown by the funnel:

![Figure 1. Levels of product customization: The three levels provide different views on the same product line variability model.](image)

*Level 1: Product derivation by suppliers.* Vendors resolve the variability captured in product line variability models to derive a product based on customers’ requirements. We have described our support for product derivation in earlier work [19, 28].

*Level 2: Product configuration by customers.* It is common in many domains that customers are not end users but are in charge of accompanying the introduction of the product. Customers further adapt the product to organizational specifics. For example, they adjust product features to accommodate tasks and roles in different departments.

*Level 3: Personalization by end users.* The end users of the product further adapt the product to their specific needs and wishes again using the variability models [13]. They personalize the application to their tasks and responsibilities.

Our three-level product customization approach is based on DOPLER [8], a decision-oriented approach to software product line engineering [22, 25]. DOPLER provides capabilities for domain engineering and product line variability modeling [9] as well as application engineering capabilities for deriving customized products using these models [19].

2.1. Product line variability model

Figure 2 depicts the key elements of DOPLER variability models:

![Figure 2. Key elements of a product line variability model.](image)

*Assets* represent the reusable elements of a product line such as software components, elements of the documentation, test cases, or requirements. Assets can depend on each other, for example, a component may require another component to function properly. Types of assets to be modeled and dependency types are defined in a meta-model. Our approach allows defining arbitrary asset types, attributes, and dependencies to allow its use in different domains [9].

*Decisions* represent variation points in a product line variability model. They allow documenting and planning variability in domain engineering, support the guidance of users during derivation, and control the automated generation of product configurations. Decisions have a name and are represented by questions for different users of the variability model. Questions can be asked to address variability in both problem space and solution space. For instance, a supplier might have to answer the questions “Shall the Archive plug-in be delivered?” and “Shall the Scanning plug-in also be included?” These questions are phrased in solution space language and require deep technical understanding. To facilitate personalization for end users the questions in the variability model should use abstractions and concepts understandable by end users.
For instance, it is better to ask a question "How do you manage documents?" with possible answers "archive" and/or "scan" instead of asking a question about the inclusion of particular archive or scan plugins. Asking an end user about the forms of communication with customers is to prefer over a question about software libraries needed for the diverse channels. Hence, to make a decision easily understandable by end users, questions need to be provided in problem space language instead of solution space language.

Decisions depend on each other hierarchically if a decision needs to be taken before another one and logically if taking a decision changes the value or alternatives of another decision. Modeling such dependencies defines the sequence for taking and processing decisions. Decisions are related with assets by explicitly modeling inclusion conditions which define for each asset when it will be part of a product. The inclusion conditions establish trace links between user decisions and the core assets. In our example (cf. Figure 3), the inclusion condition of the Archive plug-in is defined as contains(documentManagement, {archive}) while for the Scanning plug-in it is defined as contains (documentManagement, {scan new}). If the user selects “scan new” not only the Scanning plug-in but also the Archive plug-in is selected (because Scanning requires Archive).

Views help dealing with the complexity of modeling. Variability models can easily contain thousands of assets and hundreds of decisions with thousands of often non-trivial dependencies among them. Our approach allows creating views on large decision spaces to structure the provided variability during product derivation. Decisions can be grouped in views and several decisions can be treated as one entity. Grouped decisions can also be prioritized within views. In our example (cf. Figure 3), the view ClaimsManagementView groups decisions relevant for claims agents while the view MailView groups decisions relevant for mail agents. Decisions may be
part of more than one view, i.e., the ClaimsManagementView also groups decisions relevant for the MailView.

Roles allow defining responsibilities for views. For instance, the role MailAgent can see decisions in the MailView but not in the ClaimsManagementView which is only visible for a ClaimsAgent.

Users are different people responsible for taking decisions. They have a default role and can be assigned multiple roles. Switching of roles is possible. Roles define the views visible to users which in turn define the decisions a user is allowed to take. In our example (cf. Figure 3), user Jane can take on the roles of a MailAgent or a ClaimsAgent which makes the ClaimsManagementView and the MailView visible for her. User Joe on the other hand is assigned the role MailAgent and can only work with decisions visible in the MailView.

The presented elements are needed for supporting the three levels of customization of our approach shown in Figure 1: The supplier creates variability models with assets and decisions. Based on these models, the supplier can derive a product for a particular customer type. The supplier further defines views, roles, and users for its specific customers. Based on these views and roles, customers can further configure their product and define views, roles, and users for the end users. End users get a reduced view on variability that can still be resolved to personalize the product.

2.2. Tool support

We have developed three integrated tools to support our decision-oriented product line approach:

<table>
<thead>
<tr>
<th>Level</th>
<th>Tool</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1: Derivation (Supplier)</td>
<td>DecisionKing</td>
<td>create product line (model assets and decisions)</td>
</tr>
<tr>
<td>L2: Configuration (Customer)</td>
<td>ProjectKing</td>
<td>create views/roles/users for customer, derive product for customer</td>
</tr>
<tr>
<td>L3: Personalization (End user)</td>
<td>ConfigurationWizard</td>
<td>create views/roles/users for end users, configure product for end users, personalize product</td>
</tr>
</tbody>
</table>

DecisionKing [10] supports variability modeling and management. The tool allows modeling assets and decisions with their attributes and dependencies. It can be parameterized by creating a meta-model defining the assets to be modeled and their possible dependencies. This allows creating variability models for arbitrary domains. Users of DecisionKing use a high-level rule language based on JBOSS Rules¹ to define dependencies between decisions and assets. DecisionKing is based on a flexible plug-in architecture that allows extending it with arbitrary company-specific tools.

ProjectKing [19] supports preparing and guiding product derivation and customization. Based on the assets and decisions created with DecisionKing, the user of ProjectKing can define views, roles, and users. The tool also allows defining default answers for decisions. Furthermore, meta-information and recommendations on decisions can be modeled using multimedia elements such as audio or video files to provide further details and rationale for taking decisions.

ConfigurationWizard [18] supports taking decisions in product derivation and customization. It displays the decisions’ questions for users based on their assigned roles and views. By default, ConfigurationWizard displays decisions as a flat list (cf. Figure 5). However, decisions can also be visualized in graphs and trees (based on decisions’ dependencies) to ease navigation in complex variability models [20]. ConfigurationWizard is based on a flexible plug-in architecture that allows the integration of domain-specific configuration generators. For our feasibility study described in the next section we have developed such a generator capable of adapting a plug-in-based system at run-time to support personalization [28].

3. Feasibility Study

We have conducted a feasibility study in collaboration with our industrial partner BMD Systemhaus GmbH², a medium-sized company offering enterprise software products to 18,400 customers and 45,000 active users mainly in Austria, Germany, and Hungary. BMD Software is a comprehensive suite of enterprise applications for customer relationship management (CRM), accounting, cost accounting, payroll, enterprise resource planning, as well as production planning and control. BMD's target market is fairly diversified, ranging from small tax counselors to medium-sized auditing firms or large corporations. Customized products are an essential part of BMD's marketing strategy to address the needs of those markets. We did not perform a full case study involving users on all three levels. The goal of the study was to

¹ http://www.jboss.com/products/rules
² http://www.bmd.com
determine the feasibility from a technical perspective and from a modeling perspective.

3.1. Industrial challenge

BMD’s software has evolved over time to its current state but was originally not created as a product line. As typical for many complex software systems it supports product customization on different levels through different but technically not fully related mechanisms:

At the supplier level, BMD has structured the software into seven solutions which can be individually licensed and composed into five main products covering major markets. For example, BMD-Consult is optimized for chartered accountants while BMD-Commerce is targeted at corporations. However, all binaries are shipped for each product regardless of the licensed features. An individual license key activates licensed features. Unlicensed features are blocked, i.e., corresponding widgets in the user interface are disabled or hidden.

At the customer level, configuration is accomplished in a similar fashion through permissions. A customer can build individual feature subsets for different departments by revoking permissions for unneeded features. Features for which a user lacks privileges can thus be hidden.

At the end user level, the permission mechanism can also be applied to individual user accounts. A user account can be granted permissions to individual feature sets. However, since in practice this is typically done also by administrators, end users have only limited ways to personalize their application.

BMD’s customization approach of deploying the full feature set and simply hiding unused features led to several problems: The huge size of the application executable (currently around 90 megabytes regardless of how many features are used) leads to a high network load and constitutes problems for customers without a broadband connection when deploying patches over the internet. When multiple users use the same deployed application patching becomes tricky. If one user urges a patch other users are often worried about applying the patch now, as it might break other features. Finally, end user personalization by disabling features in the user account is cumbersome when end users want to perform adaptations frequently, e.g., to specific working situations, several times a day.

3.2. Conversion of the legacy software

The goal of our feasibility study was to validate the multi-level customization approach using a real-world system. BMD’s software has a total size of 4 million LOC. To test our approach, we decided to first improve the customizability of one significant subsystem of BMD’s software. We chose BMDCRM which has a total size of about 890 KLOC (420 KLOC are specific to BMDCRM, 470 KLOC are framework code used by all of BMD’s solutions). We conducted the following activities:

Decomposition of the legacy software. We first had to decompose the monolithic legacy software into a small core system and a set of pluggable extensions [28] as a preparatory step:

Definition of user-visible features. Each extension was defined to contain a single user-visible feature which can be integrated with the core system using plug-in techniques.

Identification of resources related to features. We identified artifacts such as source code and resources related to a feature to decompose features to the granularity of plug-ins.

Reengineering of identified resources to plug-ins. We then reengineered the individual components to allow their use within the plug-in platform and developed a core application such that plug-ins can be integrated. The decomposition resulted in a plug-in solution comprising 20 specific plug-ins and 28 components for the core system.

Variability modeling. We applied our product line approach and modeled the variability of the plug-in-based BMDCRM system to understand and document the possible combinations of the plug-ins and to support 3-level product customization.

Identification of assets to be modeled. We identified the following assets to be modeled on three levels of granularity (cf. Figure 3):

- Plug-ins represent single user-visible features that can be integrated individually into the application.
- Packages combine tightly related features that are commonly used together into groups.
- Solutions combine packages into a solution. For our feasibility study we only modeled the solution BMDCRM.

Definition of possible dependencies. We defined three possible dependencies between assets:

- Solution contains Package (for instance, the solution BMDCRM contains the packages Label Printing, Standard Letter, Organizer, and Docs).
- Package contains Plug-in (for example, the package Docs contains the plug-ins Archive, Scanning, Retrieval, and Import).
- Plug-in requires Plug-in (i.e., the plug-in Scanning providing document scanning features functionally requires the plug-in Archive...
providing document archiving features; this means that Archive can be used without Scanning but Scanning requires Archive).

**Specification of decisions.** We modeled possible adaptations as decisions for the three levels of customization. Decisions at higher levels constitute abstractions of decisions at lower levels. The relations between decisions and assets are described using inclusion conditions (cf. Section 2.1) that determine the set of required solutions, packages, and plug-ins as soon as decisions are taken. Please note that the mapping between decisions and plug-in assets is not necessarily 1:1. Taking a decision can include/exclude several plug-ins at once.

Overall, for BMDCRM, the variability model developed in the feasibility study contains 7 decisions, 20 plug-ins, 4 packages, and 1 solution. Figure 3 partly depicts this variability model.

### 3.3. Applying the customization approach

The following scenario illustrates how our tool-supported approach supports customizing BMDCRM according to the three levels of customization:

**Level 1: Product derivation by suppliers.** Using ConfigurationWizard users can derive a product for a customer based on the product line variability model created with DecisionKing. The partial example depicted in Figure 5 shows two decisions: “Do you want to derive a full solution or choose particular packages?” and “Which solution do you want to derive?” A user might choose to derive the BMDCRM solution which configures the default BMDCRM product (top right in Figure 5) based on the underlying variability model. All subordinate lower level decisions are determined automatically and a list of required plug-ins is computed. These can be bundled with the plug-in run-time environment and shipped as a product to customers.

The BMD user can use ProjectKing to define views on the variability models, roles responsible for these views, and users which can be assigned the roles. This determines which decisions remain unanswered and can be taken by customers later. The customer is not allowed to change the chosen solution but can only select additionally available packages and plug-ins not included in the solution by default.

**Level 2: Product configuration by customers.** The binaries of the BMDCRM solution can be deployed to the customer together with the BMD variability model. Decisions taken at the supplier level constrain the decision space for customers using the variability model to configure products for internal use. The customer further configures the BMDCRM solution based on the variability model using ConfigurationWizard. Customers can choose additional packages and configure chosen packages by taking decision "What are your tasks?" (cf. Figure 5). For example, the sales department typically answers “Customer Management” or “Claims Management” which includes packages from the BMDCRM solution. ConfigurationWizard computes a list of required plug-ins for each internal product based on the taken decisions. These plug-ins constitute the department's customized product and are deployed to individual department locations.

The customer uses ProjectKing to define the views, roles, and users for end users. Thereby the customer specifies which decisions can be taken by which end users to further customize the solution to their personal needs.

**Level 3: Personalization by end users.** End users use an already pre-configured application. However, the application still allows personalization by end users using the ConfigurationWizard. The tool utilizes the variability model and the views, roles, and users defined by the customer. The end user level comprises two aspects of personalization:

**Adapting application environments for individual users.** Different users have different tasks. For example, a user Joe in the sales department might be responsible for incoming mail orders and take the decision "What are your tasks?" by answering “Document Management”. For the subsequent decision "How do you manage documents?" he selects scanning and archiving incoming letters to customize the document management features.

**Ad-hoc adaptation to working situation.** User Jane might decide her responsibilities on certain occasions more flexibly. Most of the time she is responsible for incoming mails just like Joe. But on some days she might need to step in for a colleague who is a claims agent. Since most of the time she does not deal with claims she does not need these features constantly present. In the example shown in Figure 5, Jane takes decisions to adapt the BMDCRM solution to her needs regarding claims management. Taking decisions automatically leads to run-time adaptation of the BMDCRM system [28]. For example, Jane answers the question “How are you communicating with customers?” with “via phone” which includes phone tools in BMDCRM. She answers the decision question “Which additional capabilities do you need?” with “journal”, “memos”, and “task manager” to denote that she needs to write a journal as well as memos and that she needs to manage tasks for claims management. Jane answers the question “How do you manage documents?” with “read existing” as she does not need to archive, scan, and/or import docu-
ments as a claims agent. Based on Jane’s answers, the required plug-ins are automatically selected and BMDCRM adapts at run-time. Whenever Jane changes her working situation from mails to claims she now just has to switch her role. Already taken decisions are stored.

3.4. Results

Even though in the feasibility study we only focused on one of BMD’s solutions, we were able to test the feasibility of our tool-supported approach. Our three-level customization approach improves customizability on all three levels from product derivation down to end user personalization.

On the supplier level BMD can use the new technology to generate the same solution as before, but additionally diverse combinations of plug-ins can now be derived from the product line by taking decisions. Decomposing the monolithic application re-

Figure 5. Three-level product customization of the BMDCRM system with ConfigurationWizard based on the product line variability model (cf. Figure 3).
sults in smaller binaries to deploy; patches are much smaller and can be selectively deployed for affected components. When comparing that to the existing way of customizing we achieved a substantial improvement: Where in the prior approach the deployed binary used to be about 90 megabytes in size, binaries now are just 25 megabytes, i.e., for the BMDCRM solution. Each additional solution adds another 5-15 megabytes. Patches also got a lot lighter since individual plug-ins now range from less than 100 kilobytes to a maximum of 2 megabytes for large framework components.

At the customer level the configuration process is easier because of the tool support. Different combinations of packages and plug-ins can easily be selected by answering the questions presented with ConfigurationWizard.

At the end user level, where end users before had to use a preference dialog in a user manager to manually activate or deactivate single features, they now can take high-level decisions with ConfigurationWizard to adapt the application for the working situation at hand. And where the earlier solution required the application to be restarted to conduct the required adaptation, the application now instantaneously adapts on-the-fly at run-time [28].

We have experienced that the amount of variability resolved by different users at different customization levels can differ from domain to domain. The amount of variability resolved per level of customization is thus variable. BMD does not resolve a lot of variability beforehand but delivers a highly customizable product to its customers who then configure the product. In another project in the area of process automation in steel plants [12] we have seen that most of the variability is resolved by the supplier and only minor configuration is done by customers.

4. Related Work

We focus our discussion of related work on product derivation, configuration and personalization for end users, as well as run-time adaptation in SPLE.


Configuration and personalization for end users. While product line approaches do typically not emphasize personalization by end users this idea itself is not novel. Some software producers already go beyond providing basic installation wizards and provide more sophisticated support for end users. For example, the enterprise business application SAP/R3\footnote{http://www.sap.com} can be delivered with product configurators that allow users adapting the application for their specific needs. Another example is the SuSE Studio configurator\footnote{http://studio.suse.com} allowing end users to construct a customized Linux distribution themselves using a web-based front-end. Such configurators are however not based on variability models that can be used at multiple levels during the customization process. Personalization aims at providing users with applications customized to their very specific needs and adapting "on-the-fly" if their needs change. For instance, e-commerce applications can adapt themselves automatically based on acquired user information to provide personalized services [1]. In contrast to such knowledge-based approaches we do not acquire the required information automatically but present choices to end users based on product line variability models.

Run-time adaptation in SPLE. Diverse researchers in different areas have developed approaches and tools contributing to the run-time adaptation of systems. However, only few approaches combine software product lines and run-time adaptation: Lee and Kang [16] propose a feature-oriented approach for dealing with run-time adaptation. In their approach, reconfiguration is based on identifying binding units in feature models. The authors do however only describe conceptual support for a reconfiguration tool with no actual implementation. Wang et al. [27] describe an approach based on patterns and rules to privacy that can be used to support feature adaptation of web applications at run-time. They also describe a prototypic implementation within the ArchStudio product line architecture tool. In [14] Hallsteinsen et al. present the MADAM approach which uses variability models to describe the choices for run-time
adaptation of component-based architectures. The goal of MADAM is to support adaptation of mobile devices to changing environmental conditions such as available bandwidth or network connectivity. Their variability models therefore define choices based on sensed context information. Decisions are local to particular components and not stored in a complete decision model as in our case. While MADAM is context-centered our approach leaves the decisions to users at different levels.

5. Conclusions and Future Work

We presented a decision-oriented product line approach providing support for product customization at three levels: At level 1 suppliers use variability models to derive products from a product line customized to the requirements of a particular customer. At level 2 customers refine the configuration of the product to address organization-specific aspects constrained by the variability model used by the supplier before. At level 3 end users personalize the product, again using the same variability model. Moving from one level to another means resolving more variability. The approach is supported with the tools DecisionKing [10] for creating variability models, ProjectKing [19] for defining views, roles, and users, and ConfigurationWizard [18] for taking decisions.

Future work will focus on several aspects:

The paper presented a feasibility study of applying our tool-supported approach in the ERP domain together with our industrial partner BMD. In this study, we have also developed run-time adaptation capabilities that allow run-time personalization by taking decisions [28]. We have not yet conducted interviews and collected data from end users, customers, and suppliers to measure their acceptance regarding the utility and usability of our approach. We plan to apply our three-level customization approach in additional studies to validate its usefulness in other domains and to measure user satisfaction. Also, we will convert additional BMD solutions to a product line to further test our approach and tools.

End user personalization results in additional challenges for user support which we will investigate in future work. In case of problems end users seek for support, e.g., by contacting a help desk. Help desk staff needs capabilities to reproduce the exact configuration of the end user. As every customization decision is stored in the underlying variability model in our approach we can utilize the model for exactly that purpose. We have been developing an initial prototype integrated with ConfigurationWizard that allows end users to send requests to help desk staff over network. The help desk staff can then reproduce the

user’s configuration to handle the request. If a particular feature would be required to solve the user’s problem and that feature is not enabled, support can simply activate the respective plug-ins.

An interesting idea for future work is to use our decision-oriented customization approach to quickly develop software prototypes. Based on a variability model users could experiment with arbitrary combinations of features and different variants of the system under development.

Another important area of future research is support for product line evolution. In our feasibility study we assumed that the desired features are known and already available. It will happen however frequently in practice that yet-to-be-developed features might be requested on all three levels. We have been developing tools to integrate requirements management and product line capabilities in the past [18]. We have also developed an approach supporting the evolution of product line models [11]. Despite these initial steps more research is needed on how to deal with changes and evolution in a product line context.

Acknowledgements

This work has been conducted in cooperation with BMD Systemhaus GmbH and has been supported by the Christian Doppler Forschungsgesellschaft, Austria. We also want to thank Iris Groher for valuable feedback on earlier versions of this paper.

References