Abstract: Configuration has a number of related topics such as mass customization, software product lines, design, planning, recommender systems, software configuration, and product data management. For a discussion of mass customization, refer to Chapter 9; a discussion of software product lines can be found in Chapter 6. The other topics are discussed in this chapter.

3.1 Design

Sabin and Weigel (1998) define configuration as “a special case of design activity where the artifact being configured is assembled from instances of a fixed set of well defined component types which can be composed conforming to a set of constraints.” Following the characterization of synthesis tasks (Brown and Chandrasekaran, 1989), configuration is often categorized as routine design (Günter and Kühn, 1999; Stumptner, 1997; see Figure 3.1).

Brown and Chandrasekaran (1989) describe routine design (design class 3) as a problem, where the specifications of objects, their properties, and compositional structures are already given, and the discovery of a solution is based on a known strategy. This characterization is similar to the definition of Sabin and Weigel (1998). An example of routine design is the quote generation for technical systems on the basis of a complete description of components, their properties, a compositional structure, and related constraints. If the limitations of routine design are not given in the domain, other possible categorizations are innovative design (design class 2) and creative design (design class 1 on the top of the scale). An example of innovative design is the creation of an upgrade version of a basic mobile phone type that—compared to the old model—includes additional features. An example of creative design is the artistic design of a new furniture line. From these examples, we can infer that transitions between design classes are smooth and subcategorizations can be introduced (Hotz and Vietze, 1995). For a detailed discussion of design tasks, refer to Brown and Chandrasekaran (1989).
Different types of design tasks: (1) designs are derived from a fixed set of component types and related constraints (routine), (2) new components and constraints can be included within the scope of the design process (innovative), and (3) components and constraints are designed from scratch (creative). inc denotes incompatibility constraints.

Representing a planning task as a configuration task.

### 3.2 Planning

Planning can be defined as a process of sequencing a set of activities in such a way that a defined goal can be accomplished. Thus, planning deals with the composition of actions whereas configuration deals with the composition of components. Both tasks are similar\(^1\) with regard to the calculated results and the used problem solving approach. An example of how to translate a planning task into a corresponding configuration task is depicted in Figure 3.2.

Solution spaces are in both cases restricted by constraints: in planning between actions and in configuration between components. A distinction between planning and configuration is the influence of temporal restrictions in planning situations (e.g., action a1 before a2 or a1 and a2 at the same time). The question “does a condition still hold after applying a planning operator?” is a main concern in planning tasks. It introduces additional complexity, which explains research interests in approaches that exploit elaborated heuristic search methods (Biundo et al., 1993; Edelkamp and Schrödl, 2012; Görz, 1993).

---

\(^{1}\)See also www.puk-workshop.de.
3.3 Recommender Systems

Recommender systems (Felfernig et al., 2007, 2013; Jannach et al., 2010; Ricci et al., 2011) support users in the process of finding and selecting products (items) from a given assortment. Examples of such items are movies, books, songs, financial services, apartments, and digital cameras. Configurators are often built with a similar goal in mind. However, a major difference between product configurators and recommender systems is the way in which product knowledge is represented (Falkner et al., 2011). Where configurators often operate on a configuration knowledge base, recommender systems typically operate on a table of explicitly defined solution alternatives. A reason for using a knowledge base is that the space of possible solutions makes an explicit representation impossible. The major commonality between recommenders and configurators is that both systems try to achieve the goal of proactively supporting users in finding a solution (configuration) that fits their wishes and needs. There are three basic approaches to the recommendation of items.

First, collaborative filtering (CF; Konstan et al., 1997) is based on the idea of word of mouth promotion. Such systems determine recommendations by identifying so-called nearest neighbors with a similar rating behavior (preference profile) compared to the current user. On the basis of this information, items are recommended that are not known to the current user. Second, content-based filtering (CBF; Pazzani and Billsus, 1997) is based on the idea of recommending items that are determined on the basis of the similarity between the preferences of the current user (properties of items already investigated by the user) and item properties extracted from item descriptions. An example of the application of content-based recommendation is the recommendation of web sites. Third, knowledge-based recommendation (KBR; Felfernig and Burke, 2008; Felfernig et al., 2006) recommends items on the basis of a predefined set of constraints (rules) and/or similarity metrics. The major difference compared to the approaches of CF and CBF is that in KBR deep knowledge about the product assortment is needed in order to be able to determine recommendations (this is not the case with CF and CBF approaches). For a more detailed discussion of recommendation technologies refer, for example, to Jannach et al. (2010) and Ricci et al. (2011).

| table 3.1 | An example session $\times$ item matrix for collaborative feature recommendation (Falkner et al., 2011). The values of $s_k \times f_i$ denote the order in which features $f_i$ have been specified by a user in session $s_k$. |
| session | $f_1$ | $f_2$ | $f_3$ | $f_4$ | $f_5$ | $f_6$ |
| $s_1$ | 1 | 0 | 3 | 2 | 0 | 4 |
| $s_2$ | 1 | 0 | 0 | 2 | 4 | 3 |
| $s_3$ | 2 | 3 | 0 | 1 | 4 | 0 |
| $s_4$ | 1 | 4 | 0 | 2 | 0 | 3 |
| $s_5$ | 1 | 0 | 0 | 2 | 0 | 0 |
Since we can observe an increasing demand for functionalities that proactively support configurator users, there is plenty of potential to exploit recommendation technologies to support the interaction with configuration systems (Falkner et al., 2011; Tiihonen and Felfernig, 2010). One such functionality is the recommendation of features. For example, due to the large number of questions, a recommender can act as a preselector of questions that should be posed to the user (see Table 3.1). Collaboratively selecting features is one approach to predict features that will be of interest for the user. This kind of recommendation can be implemented on the basis of collaborative filtering (Konstan et al., 1997). If we assume that a user in his/her current session has already selected and specified the features \( f_1 \) and \( f_4 \), the most similar sessions (the four nearest neighbors) are the sessions \( \{s_1, s_2, s_4, s_5\} \) and \( f_6 \) would be recommended as the next text feature to be specified by the user (\( f_6 \) is the feature that has been selected by the majority of nearest neighbors). Other examples are the recommendation of feature values (Falkner et al., 2011; Tiihonen and Felfernig, 2010) and the recommendation of repair alternatives for inconsistent requirements (Felfernig et al., 2009).

### 3.4 Software Configuration and Version Management

Configuration technologies originally evolved from hardware configuration (McDermott, 1982). Configuration can also be applied to software and software-intensive systems, due to the generality of the developed technologies. Software-intensive domains such as Car Periphery Supervision (CPS) require the combination of hardware and software systems. CPS systems monitor the local environment of a car on the basis of sensors installed around a vehicle. The measurement and evaluation of sensor data enables different kinds of applications. These can be grouped into safety-related applications such as pre-crash detection, blind spot detection, and adaptive control of airbags and seat belt tensioners; and comfort-related applications such as parking assistance and adaptive cruise control (Thiel et al., 2001). Due to the variety of involved hardware and software components and a high variety in the set of possible customer requirements, CPS becomes an interesting area for applying configuration methods (Geyer, 2002; Hotz et al., 2006). Examples of the application of configuration technologies for “pure” software configuration are discussed in Hotz et al. (2006), Männistö et al. (2001), Myllärniemi et al. (2005), and Tiihonen et al. (1998). Approaches to the configuration of software product lines (based on feature models) are discussed in Chapter 6.

Software Configuration Management (SCM) handles dependencies of software artifacts in the context of software development projects. In the majority of the cases, these artifacts are represented as files (see, e.g., Ylinen et al., 2002, for configuring Linux systems). Major functionalities are the creation of a new version of a software artifact, the restoration of partial (or complete) older versions, the merging of different versions, and notification services that keep developers informed about changes in the repository. As such, SCM is primarily applied in the context of software development processes whereas software configuration technologies focus more on the variability management of stable software artifacts. A major difference compared to configuration technologies is the lack of an abstract, declarative model of the source code being configured (Männistö and Sulonen, 1999). Knowledge-based systems (such as configuration systems) provide a more formal notion of consistency and completion than SCM systems (Tiihonen et al., 1998) and can give support in this issue. Further aspects related to Software Configuration Management are discussed in van der Hoek et al. (1995).
3.5 Product Data Management

Product Data Management (PDM) supports the definition and maintenance of information required to design, manufacture, and maintain products. Examples are specifications related to the manufacturing process, technical product data, and specifications of material needed for production purposes. Product parts stored in a PDM system are typically characterized with attributes such as part identifier, name of the supplier, price, and CAD drawings. PDM systems may be used to store configurations, derived documents, and bills of material created with configurators. Integration of a PDM system and a configurator for storing configurations can be realized directly or via an ERP system. A configurator and a PDM system often share some common product data: in the context of the working example (see Chapter 6), current hard disk and processor types could be added to the configuration model through PDM integration. PDM systems increasingly include facilities for configuring products, especially from the production configuration point of view. In this role, PDM systems can act as a kind of corporate knowledge repository that supports information interchange among users from, for example, marketing, sales, product development, production, and the customer.

3.6 Conclusion

In this chapter, we briefly discussed topics that are directly related to configuration. Distinctive features between configuration and these related topics are reasoning about temporal dependencies in planning, open configuration models in design, explicit representation of alternatives in recommender systems, explicit handling of dependencies in software configuration management, and flexible data management in product data management systems.

References


Product Data Management nowadays is often discussed in context of the disciplines of Product Lifecycle Management (PLM) and Master Data Management (MDM). Discussing the relationship is outside the scope of this book. See, for example, White and Halpern (2009).