Integrated Design and Process Technology, **IDPT-2002**
Printed in the United States of America, June, 2002

EXTENDING FEATURE DIAGRAMS WITH UML MULTIPLICITIES

Matthias Riebisch, Kai Böllert, Detlef Streitferdt, Ilka Philippow

Ilmenau Technical University, Max-Planck-Ring 14, P.O.Box 100565;
98684 Ilmenau, Germany
{Matthias.Riebisch|Detlef.Streitferdt|Ilka.Philippow}@TU-Ilmenau.DE

ABSTRACT

Feature diagrams are an important product of domain analysis for product lines or system families, respectively. They describe relations between requirements and distinguish between common and variable characteristics. Feature diagrams, as part of the feature model, form the basis for configuring the system. Current principles do not supply a complete description of the semantics of relationships and dependencies between features. Thus, the development of methods and tools for elaborating configurations is not possible. This paper presents some enhancements to feature diagrams. In addition the paper deals with the inclusion of feature models in the development process of product lines.

Index terms: domain analysis, software product lines, system families, software reuse, feature modeling

INTRODUCTION

Software reuse employed to many applications leads to economic advantages. By reusing once developed assets, development costs will be reduced. In addition a faster development cycle is possible when reusing existing assets. Last but not least the maturity level of frequently used software will be higher due to continuous bug fixes applied to the software. Since the middle of the 80th the main objective of software engineering is software reuse (Tracz, 1987).

Multiple paradigms of software reuse at code and design level have been developed. Many efforts were taken to reduce the number of systems developed from scratch in favor of reusable software assets. Here software engineers realized that communication aspects and the motivation of the organization as well as the management need to be considered, to develop reusable assets successfully. Reusable assets are developed using domain analysis techniques, to handle and make use of similarities while inside the requirements analysis phase. Product lines or system families, respectively, aim towards a group of software systems with similar requirements. The development of a common and reusable core for all systems is addressed by this development strategy as well as economic issues of the development itself. Investments, profitability and organizing a software company are considered through comparison with other economic areas.

Product line development starts with modeling the common and variable requirements of a group of software systems. Requirements will be related to the hierarchically organized features. Setting up rules and relationships within the feature model reduces the possible choice of features. The developer decides which parts of the system will be realized as common and which will be realized as variable parts based on the feature model.

Within the configuration step a system will be composed out of common and variable elements. A developer will choose out of the available features. The outcome is a configuration describing a specific system of the product line. A feature is related to a set of requirements as well as to corresponding parts of the system architecture, to ensure a smooth development throughout all development phases. Tool-based creation of configurations requires clearly defined relations inside the feature model.

The advantage of product line development methods is the development of many systems based on the common architecture. Such a development needs to apply the methods of product line development in each development phase. The requirements model will have to be separated into common and variable requirements. The transformation of a requirements model into a feature model already reveals variants of the product line, which can be used for a prediction of further development tendencies. Creation of a far reaching project plan and the evolutionary development of a product line is now possible (Clements et al., 2002; Riebisch et al. 1999). Feature models form an important foundation for the development of product lines and contribute to the success of the product line.

STATE OF THE ART

Feature modeling focuses on the hierarchical structuring of requirements elicited out of a given problem domain. An important method is Feature-Oriented Domain Analysis (FODA) (Kang et al., 1990). The feature diagram (Fig 1) represents the relation between features and the corresponding requirements. Czarnecki (2000) used features for Generative Programming within requirements analysis and developed an extension for feature diagrams.

All hierarchically organized feature diagrams start with a concept node at the root position. The concept refers to a property, a product or a domain. At the next level beneath the concept features will follow hierarchically. Besides the

normal hierarchies, Czarnecki (2000) also allows directed graphs, which causes valid relations close to inheritance in object-oriented languages. The relation of a feature to the product line is expressed using one of the specifiers *mandatory*, *alternative* or *optional*. This establishes the reference to commonalities and variabilities of similar systems and thus to product lines.

Non-chooseable features are marked as mandatory. These features will be present in all members of the product line. A filled circle at the top of the feature identifies a mandatory feature. Optional features are only present in an application if the customer has chosen them. An empty circle at the top of the feature identifies an optional feature. A path starting at the concept leading down to the leaves of the tree and containing just mandatory features is part of the core of the product line. If there is an optional feature part of the path, then just the path above this optional feature up to the concept is part of the core.

The *excludes* and *requires* relationships between features enable the expression of additional constraints and dependencies. Frequently, they are shown graphically by dashed arrows between features with type descriptors <<requires>> or <<excludes>> very similar to UML stereotypes (Rumbaugh et al., 1999).

SETS OF ALTERNATIVE FEATURES

A segment of a circle between the outer edges of a set of features denotes a choice out of a set of optional features. Czarnecki et al. (2000) represents the logical OR-relation by a filled segment. Here we can choose at least one out of the specified set of features. Features may but do not have to be part of the choice. Empty segments or arcs represent alternative choices. Just one out of a set is a possible choice. Out of all possible combinations of optional and mandatory sets of features with OR-relations and alternatives we have chosen a relevant subset for the paper as shown in Fig 3.

The three diagrams forming the top line of Fig 3 refer to the base types used in feature diagrams. In diagram (1) features B, C and D have to be chosen. The alternative choice in diagram (2) requires exactly and not more than one feature out of the set B, C and D. Finally in diagram (3) one could choose a non-empty set out of B, C and D.

By using these concepts, cases with ambiguous semantics happen. To overcome this problem, Czarnecki (2000) proposes a normalization into an unequivocal notation, which is referred by the diagrams in the middle and bottom line of Fig 3.

Diagrams (5) and (6) require the choice in two steps. Based on the OR-relation a non-empty subset needs to be chosen. After this choice of a set as in diagram (6), or the set containing feature B in diagram (5) respectively, we can choose again, contrary to the original intention of the OR-relation. This will enable the empty set as a possible solution. This ambiguity can be removed by normalizing diagrams (5) and (6) to diagram (4).

The alternative choice in diagram (9) will lead to exactly one feature, what causes in case of choosing feature B again the empty set. This is also a choice in two steps, which could be normalized as shown in diagram (10).

Removal of ambiguities and unification of the notation requires the extension of feature diagrams.

FEATURE SPECIFIERS

If features in a feature diagram are organized as a tree, then each feature has a property that holds the feature's specifier (mandatory, optional, or alternative). This solution, "specifier is a property of a feature", is no longer valid if the features in a diagram are organized as a directed-acyclic graph, according to the definition of Czarnecki et al. (2000). Fig 2 shows an example in which feature C is a "subfeature" of feature A and B. From the point of view of A, C is mandatory. However, viewed from B, C is optional. Hence, we can no longer store a feature's specifier as a property of the feature. To circumvent this problem in both tree-based and graph-based feature diagrams, specifiers should be stored as a property of feature relations.

MULTIPLICITIES IN FEATURE DIAGRAMS

Multiplicities are a very common modeling element. They are used in UML class diagrams or in entity-relationship diagrams, to name a few (Rumbaugh et al., 1999). Assume, for instance, a class diagram in which one class is associated with another class and the association is annotated with the multiplicity "0..1". The semantic of this multiplicity is that at runtime each object of the first class may have at most one relation to an object of the second class.

Feature diagrams also have multiplicities, though they are less obvious than in UML class diagrams. In feature diagrams a multiplicity shows up whenever features are combined into a set. Fig 3 depicts all types of sets that are possible using the feature diagram notation introduced by Czarnecki (2000). The multiplicities of the sets shown are as follows:

- 0..1 — at most one feature has to be chosen from the set, (9) and (10)
- 1 — exactly one feature has to be chosen from the set, (2)
- 0..* — an arbitrary number of features (or none at all) have to be chosen from the set, (5) and (6)
- 1..* — at least one feature has to be chosen from the set, (8)

Certainly, this list of possible multiplicities in feature diagrams covers the most common cases. In practice, however, often situations arise in which a set of features has a multiplicity like "0..3", "1..3", or simply "3". Such multiplicities cannot be expressed using the current notation.

Therefore, we propose to change the notation of feature diagrams. The goals for the new notation are:

(a) To annotate multiplicities for sets of features in a more convenient, understandable way;

(b) To allow for other multiplicities besides the four common ones listed above;
(c) To unify the notation of multiplicities in feature diagrams with the multiplicity notation prevalent in the UML.

Our new notation uses the following elements:

- A feature is a node in a directed-acyclic graph. Relations between features are expressed by edges between features. A circle at the end of its corresponding edge determines the direction of a relation.
- If this circle is filled, then the relation between the features is said to be mandatory, i.e. when the feature at the relation's origin is chosen, the feature at the relation's destination has to be chosen, too. If the circle is empty, then the relation is non-mandatory, i.e. the feature at the relation's destination needs not to be chosen; it is optional.
- Optional relations that originate from the same feature node can be combined into a set. Every relation can only be part of one set. A set has a multiplicity that denotes the minimum and maximum number of features to be chosen from the set. Possible multiplicities are: 0..1, 1, 0..n, 1..n, m..n, 0..*, 1..*, m..* (m,n $\in$ N). Visually a set is shown by an arc that connects all the edges that are part of the set. The multiplicity is drawn in the center of the arc.
- Relations between features that are located in different, not adjacent parts of the graph may not be shown in feature diagrams, because this reduces the clarity of the diagrams. Instead, such relations can be described in a textual form in the feature model.

The remainder of this section compares the old with the new notation using a couple of figures. First of all, Fig 4 (1) shows a mandatory relation from A to B, C and D. The notation of this relation remains unchanged, because it is unambiguous and easy to understand. Fig 4 (2) shows the same relations as (1), but this time combined into a set of alternative features. The multiplicity of this set is "1", so the new notation of this alternative relation is (3).

Fig 5 (4) shows optional relations from A to B, C and D. The diagrams (5) and (6) express the same semantic as (4) with a set of optional OR-features. The multiplicity of all three diagrams is "0..*", so the new notation of (5) and (6) may be either drawn as in (7), which is the preferred way, or may be normalized to (4).

Fig 6 (8) shows a set of OR-relations from A to B, C and D. The multiplicity of this set is "1..*", so the new notation of (8) is (11).

Fig 7 (9) shows a set of optional alternative features B, C and D, which can be normalized to (10). The multiplicity in both diagrams is "0..1", so the new notation is (12). Fig 8 shows how the new notation might be used to express specific multiplicities.

ADDITIONAL CONSTRAINTS AND DEPENDENCIES

The relationships mentioned in the last section lead towards hierarchically structured features. In contrast, *excludes* and *requires* relations are part of this hierarchy but are not hierarchically organized themselves. They are used to formulate constraints and dependencies between any pair of features of the diagram. Given these new graphical elements, the complexity of the diagram is increased while the clarity is reduced. As a result, developers face a higher effort for program comprehension and maintenance. Thus, we propose to express constraints and dependencies in a textual and formal manner, by using a subset of UML's Object Constraint Language (OCL).

A CASE STUDY

This section describes a more complex example, very similar to one of our current projects in industry, in which the new notation of multiplicities in feature diagrams has proven to be useful.

The project started with the development of a software system for our university library. We used the UML to model the system and Java to implement it. After the system was released, we recognized that the system would be of interest not only to university libraries, but also to the local city library including their buses, which drive through the countryside, lending books to customers who live in the villages. All three libraries have the same basic requirement: They want to manage the books borrowed by their readers. Beyond that the requirements differ: The university library needs to categorize their readers because some of them, namely the employees of the university, enjoy more relaxed conditions when to return borrowed books. The city library, on the other hand, would like to daily synchronize their data with data collected by the buses during their tours. However, we did not have the resources to develop and maintain two different library systems, so we decided to build a system family based on the existing system.

Common Features

Fig 9 shows the initial version of the feature diagram for the family of library systems. The diagram contains those functionality (or features) that are common to all systems in the family: borrowing books, managing books and readers, reminding readers of overdue books, and the necessity for readers to identify themselves if they want to borrow a book. The parameter "time limit" (an integer value > 0) specifies after which period of time a reader has to return a borrowed book to the library.

Variable Features

Next the requirement engineering identified the optional feature "Reminding overdue books by e-mail". To realize

this feature technically, readers must notify the library of their e-mail address; another optional feature was found and added to the feature diagram as a subfeature of "Managing readers". The dependency between both optional features is expressed in the diagram by the «requires» relation. Fig 10 depicts the second version of the feature diagram for the family of library systems.

Finally, three features with sets were identified in the library system family (see Fig 11):

1. Different items that libraries can manage and lend to their readers
2. Different ways to identify readers when borrowing items from the library
3. Different data that libraries store about their readers in the system

First, a library system developed from the family can manage one or more of the following items: books, journals, and/or audiobooks. At least one of these items must be managed by the system. Otherwise no reader can borrow anything from the library.

Second, if a reader wants to borrow a book, he has to identify himself to the librarian. This can be done either by a chipcard or a biometric sensor (e.g. fingerprint check). A library system supports only one procedure, i.e. when developing the system one of the two alternatives has to be chosen.

Third, every time a new reader registers himself at the library, the librarian has to enter data about the reader into the system. The data can be used to authenticate the reader if he has lost his chipcard or wants to prolong a book by phone. Some of the data is optional, i.e. the reader may choose not to reveal the data to the library. Other data is required by the system. The feature „Required reader data" determines the required ones. A mandatory data item for all library systems is the reader's name. In addition, two more data items must be chosen from the set. If the system is to be built for a German library, suitable data are the reader's address and its birthday. For a library in the USA, the reader's social security number (SSN) and his mother's maiden name are more appropriate.

In these three cases, feature relations can be modeled much more straightforward with the extensions of this paper than without. Feature multiplicities - as shown in the feature set for Required Reader Data in Fig 11 (bottom) - are needed frequently when constraints and resource limits apply. The extensions lead to higher understandability; the danger of inconsistent feature relations reduces. In many cases, *requires*- and *excludes*-relationships are needed in addition to the extensions presented here. Our experience shows, that in some cases relations within the tree and *requires*-relationships can be used alternatively. However, due to complexity of industrial projects, model clearness is an important issue.

SUMMARY AND OUTLOOK

This paper introduced a new notation for feature diagrams, emphasizing the multiplicity of sets of features. Unlike the current notation, explicit modeling of multiplicity is used in our extension of feature diagrams. The annotation of the multiplicity of feature subsets is realized in BNF, which eases the acceptance for developers, who are aware of the UML. In addition, the applicability of the new notation was shown within a case study of middle-sized complexity.

Future efforts aim towards the integration of feature modeling into a CASE tool environment. Further improvement will be obtained through the use of traceability links between features and design elements as well as the implementation itself. As a result, reengineering, maintenance and automated documentation activities will be supported (Sametinger et al., 2002).

Finally, configuration tools for product lines are under development. They are making use of the new feature model with dependencies and constraints. This work is performed in cooperation with industrial partners to ensure permanent screening of the practical relevance.

REFERENCES

Clements, P.; Northrop, L., 2002, "A Framework for Software Product Line Practice. Version 3.0", Software Engineering Institute, Carnegie Mellon University, Pittsburgh. Available online at: http://www.sei.cmu.edu/plp/

Czarnecki, K., Eisenecker, U.W., 2000, "Generative Programming" Addison-Wesley, Reading, MA, USA.

Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A., 1990, "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh.

Riebisch, M., Franczyk, B., 1999, "Evolutionary Development of Frameworks – from Projects to System Families", *Proceedings IDPT-99*, Kusadasi, Turkey, Society for Design and Process Science.

Rumbaugh, J., Jacobson, I., Booch, G., 1999, "The Unified Modeling Language Reference Manual", Addison-Wesley, Reading, MA, USA.

Sametinger, J., Riebisch, M., 2002, "Evolution Support by Homogeneously Documenting Patterns, Aspects and Traces", *Proc. 6th European Conference on Software Maintenance and Reengineering*, Budapest, Hungary, March 11-13, 2002, IEEE Computer Soc. 2002, pp. 134-140

Tracz, W. (Ed.), 1987, "Software Reuse - emerging technology", Computer Society Press, 1987

FIGURES

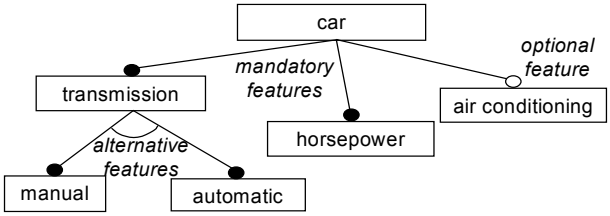

Fig 1: Example of a feature diagram (Kang et al.,1990)

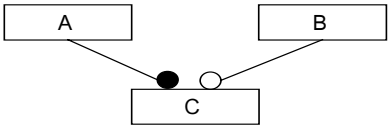

Fig 2: Relations between features in feature diagrams organized as graphs

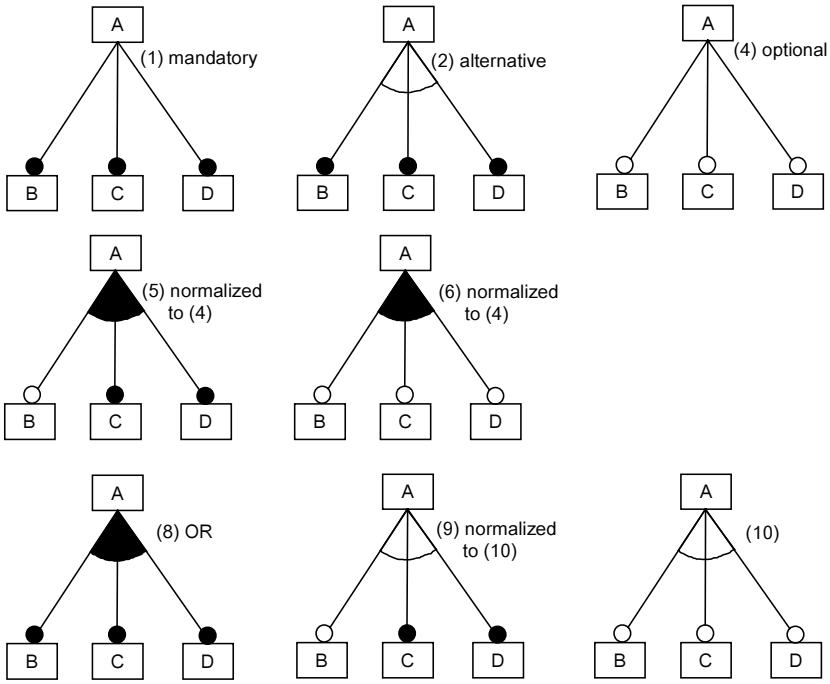

Fig 3: Relation between features according to Czarnecki et al. (2000).

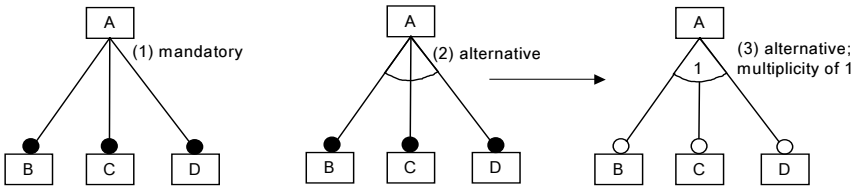

Fig 4: New notation for mandatory and alternative features

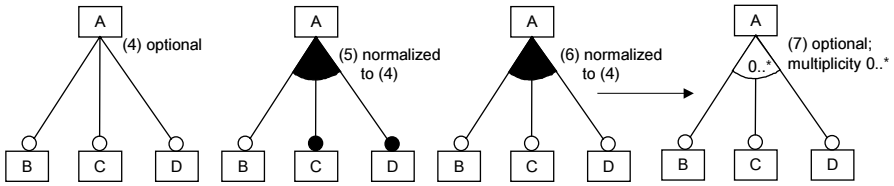

Fig 5: New notation for optional OR-features

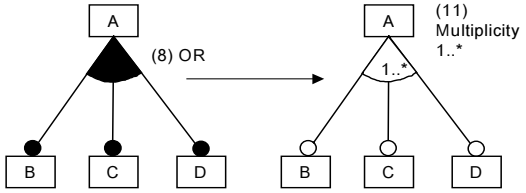

Fig 6: New notation for OR-features

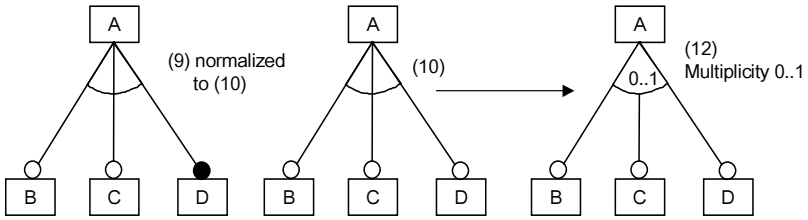

Fig 7: New notation for optional alternative features

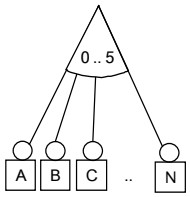

Fig 8: New notation for arbitrary multiplicities for sets of features

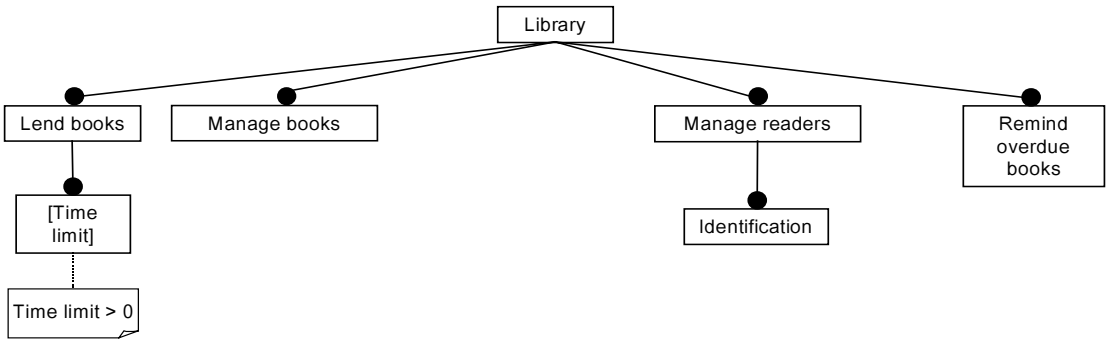

Fig 9: Common features of all library systems in the family

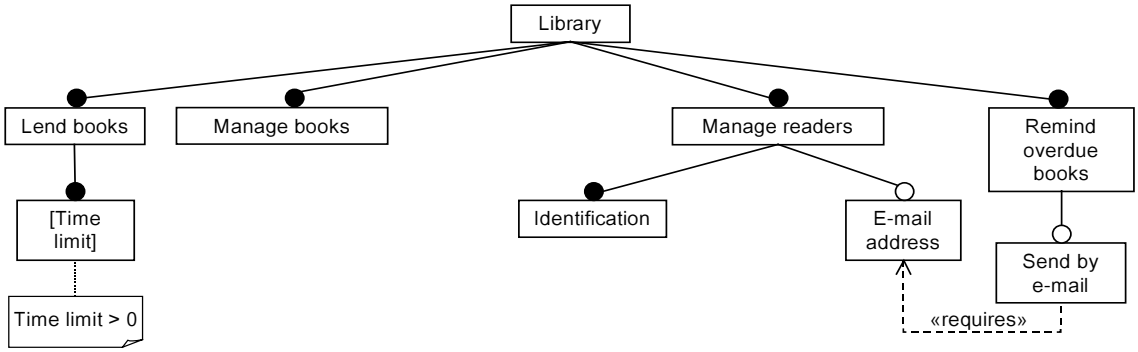

Fig 10: Optional features in the library system family

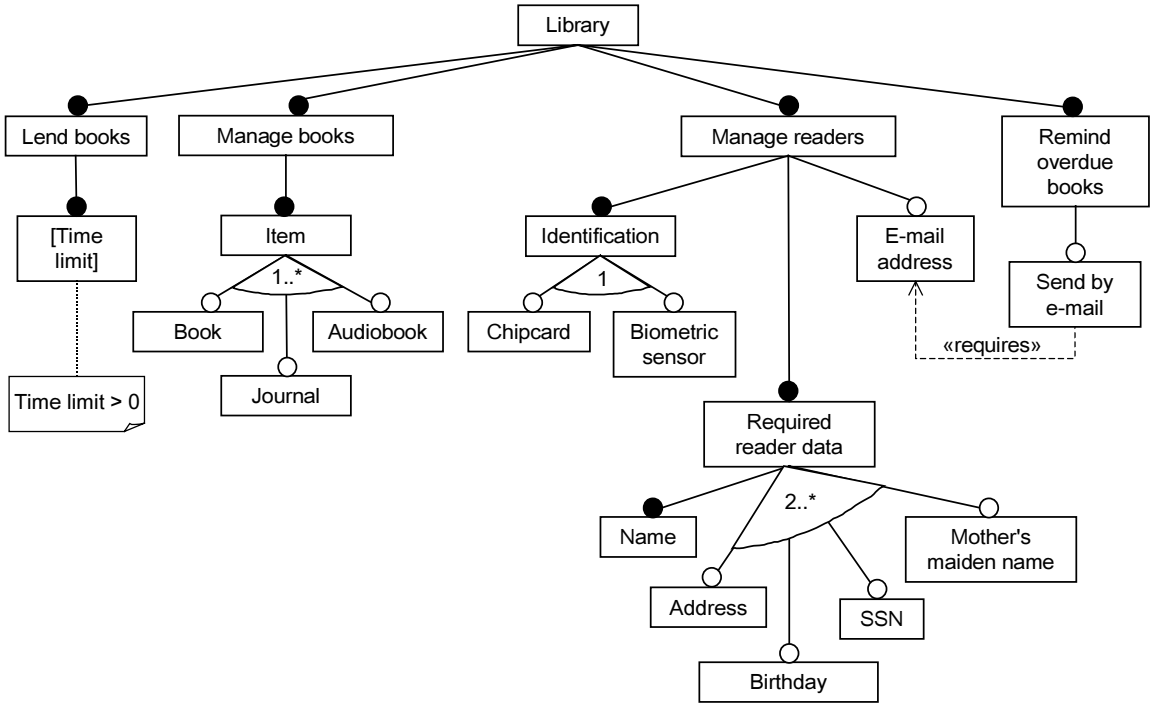

Fig 11: Feature diagram of the case study with sets of features