Position Paper:
Feature Interaction in Composed Systems

E. Pulvermüller[1] A. Speck[2] J. O. Coplien[3] M. D'Hondt[4] W. DeMeuter[4]

[1]Institute for Program Structures and Data Organization
Universität Karlsruhe, Germany.
http://i44w3.info.uni-karlsruhe.de/~ pulvermu
pulvermueller@acm.org

[2]Intershop
Jena, Germany
http://www-pu.informatik.uni-tuebingen.de/users/speck
a.speck@intershop.com

[3]Bell Laboratories Lucent,
Naperville IL, USA
http://www.bell-labs.com/~ cope/
cope@research.bell-labs.com

[4]Vrije Universiteit Brussel, Belgium
http://prog.vub.ac.be/
mjdhondt@vub.ac.be
wdmeuter@vub.ac.be

Keywords: Feature interaction, feature, feature modeling, composition

Abstract

Feature interaction is nothing new and not limited to computer science. The problem of undesirable feature interaction (feature interaction problem) has already been investigated in the telecommunication domain. Our goal is the investigation of feature interaction in component-based systems beyond telecommunication. The position paper outlines terminology definitions. It proposes a classification to compare different types of feature interaction.

A list of examples give an impression about the nature and the importance of feature interaction.

1 Introduction and Problem

The workshop "Feature Interaction in Composed Systems" deals with a problem which is not new. In fact, as opposed to that, it's a problem even older than human life.

In the domain of telecommunication this problem was explicitely explored first in the beginning 1990s using the term "feature interaction problem". In a series of workshops (the first was held in 1992) the difficulty to manage implicit and unforeseen interactions between newly inserted features and the base system have been examined.

However, the problem is not limited to the telecommunication domain. As opposed to that

feature interaction is an issue which occurs in nearly all domains although not known under the name “feature interaction”.

In the last two years we found a growing awareness of this problem in the domain of system composition far beyond telecommunication issues. With the emerge of aspect-oriented programming it has become obvious that with the growing number of system units their interaction is a problem of its own, requiring research by its own. While AOP, CF, SOP, AP [1] and related approaches reveal this problem it’s not limited to those approaches either. It already exists in object-oriented or component-oriented systems, for instance.

In many discussions we found that already a lot of work exists dealing with feature interaction in various domains and applying various programming paradigms.

The workshop aims at collecting the different problems and to provide a platform for knowledge transfer.

Some important questions are:

- What are suitable definitions for “feature” and “feature interaction”?
- Are there any commonalities in the different problems and / or their solutions?
- How can the problems be categorised in problem classes?
- What is the influence of advanced separation of concerns or component-based approaches to feature interaction problems and vice-versa?

While it is easy to compose a system technically it’s hardly explored how systems can be combined in a way that the result is a valid and also reasonable system. The developer faces a lack of clearly structured composition and connection concepts.

2 Terminology or “What is a Feature?”

When discussing about feature interaction it’s important to consider existing terminology.

In a series of workshops the feature interaction problem has already been investigated in the telecommunication domain since 1992. The workshop statement explains that “feature interaction occurs when one telecommunication feature modifies or subverts the operation of another one”.

A definition found in the telecommunication community is as follows: “The feature interaction problem can be simply defined as the *unwanted interference between features running together in a software system.*” A simple example given in [9] is a mailing list echoing all emails to all subscribers. If one of the subscribers has enabled the vacation program (without first suspending messages from the mailing list) an infinite cycle of mail messages between the mailing list and the vacation program will occur.

In [10] a feature is defined as “an extension to the basic functionality provided by a service” while a service is explained as “a core set of functionality, such as the ability to establish a connection between two parties”.

While these definitions concentrate on telecommunication, large and distributed systems or multimedia systems in particular the goal of this workshop is the investigation of feature interaction in the domain of software composition and systems built from components.

We distinguish the terms “feature”, “concern” and “requirement”, “service”, “component” and “aspect” according to our experiences as follows:

In [4] you may find the following explanation for “features”: “A feature is something especially noticeable: a prominent part or detail (a characteristic). A feature is a main or outstanding attraction, e.g. a special column or section in a newspaper or magazine”. Its origin is from Latin: “factura” which means the “act of making” or from

"facere" which means "to make, do".

According to this definition we use the term feature in a broader sense than just as an extension to some basic functionality. It's an observable and relatively closed behaviour or characteristic of a (software) part. In software, it's not just an arbitrary section in the code except this code section "makes something of outstanding attraction". This definition is fuzzy revealing the fuzzy nature of features. This viewpoint is consistent with that expressed in [2] where a feature is explained as "any part or aspect of a specification which the user perceives as having a self-contained functional role".

A feature is something an application has to do. It may be composed from other features. A basic feature should be as small as possible (basic building blocks). What a feature does should be documented. This might be done either manually (the implementer is forced to document it) or by deriving it from the program structure and program flow.

We have a distinction between problem domain features and features on the implementation level (cf. figure 1). Different alternative implementation features may realise the higher-level problem domain features. Moreover, a problem domain feature may be implemented by one or more implementation feature.

A component may have several features and, vice-versa, it realises at least one feature (otherwise the component is not useful). Moreover, a component implements functionality and has non-functional properties (e.g. real-time properties, platform). "Non-functional" is all which is not implemented directly. Therefore, a component *has* non-functional properties but not *implements* those. Non-functional properties are additional properties implicitly resulting from the code. However, a consequence is that the developer may have to implement mechanisms to check whether the required non-functional requirements are met, i.e. you may even find the non-functional properties in the code.

A feature may be implemented as functionality

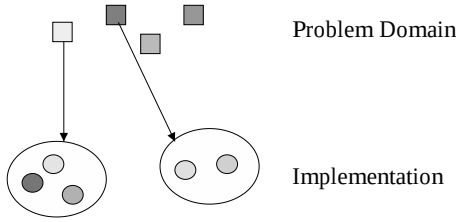

Figure 1: Levels of Features

or may have a non-functional nature. The term "feature" captures both.

A feature has the property that it is a service if it is localised in one component and if it refers to some functionality. However, a feature may be implemented in several components. In this case, the feature is (implemented) cross-cutting. This is more a question of how a feature may be implemented than a question of the nature of the feature itself. Aspects are a notation which may be used to express cross-cutting features (primarily on the implementation level). Therefore, the concept of aspects is orthogonal to the nature of features.

A requirement is something a stakeholder demands and it refers to the problem domain whereas a feature is not limited to the problem domain. A requirement may result in one or several feature(s) in the final system.

A concern is something we are concerned about (at the moment). Note that this is a temporal statement while a feature is permanent. A feature might become a concern if somebody is concerned about. On the other hand, a developer or stakeholder may be concerned about something which is not a feature. Therefore, not every concern results in a feature.

We have to distinguish between intended interactions between features, interactions between features which is not intentional but don't result in errors (or may even have positive side-effects) and unintended and undesirable feature interaction not known in advance and leading to faulty applications. Figure 2 shows a classification scheme al-

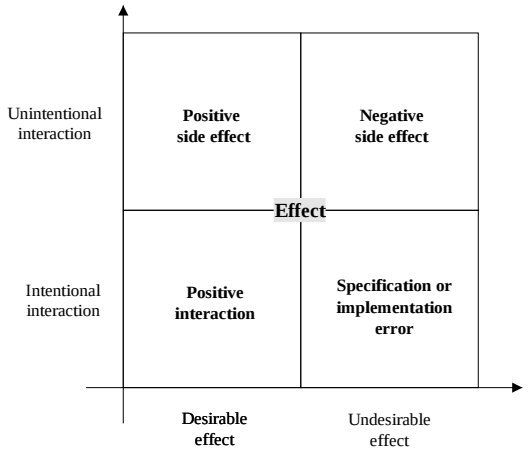

Figure 2: Feature Interaction Classification

lowing to classify, compare and assess a detected interaction.

3 Examples for Feature Interaction (Problems)

In the following some examples about undesirable or unforeseen feature interactions are listed. These examples are of different application domains giving an impression about the range of occurrences in practice.

- Example with modularised Corba functionality [5]

  In order to keep an application independent from the communication technique the communication code may be separated in aspects applying aspect-oriented programming.
  When a client wants to access a service exposed by a specific server the client has to obtain an initial reference to the server. This can be done either via a name server or via a file-based solution (the reference is stored as a string in a file which is accessible for clients and server). Aspects realising one of these two alternatives are exclusive. This is already known at design and implementation time. Let us assume that this knowledge was not captured at design time. As a consequence it might happen that the developer configures the system during the deployment phase with both mutual exclusive features. It might happen that even the compilation or weaving doesn't report this as an error. However, the running system behaves in an unforeseen way.
  An approach to deal with this problem may be found in [3]. Logical rules describe the dependencies between the aspects. During run-time pre- and post-conditions assure that these logical rules are not violated.

- Telecommunication

  Feature interaction is a typical problem in the telecommunication domain. Due to high competition and market demand telecommunication companies are urged to realise a rapid development and deployment of features.

  A list of features in the telecommunication domain may be found at [8]. Examples of features are forwarding calls, placing callers on hold, or blocking calls. It's obvious that some of the features lead to effects which are unforeseen if combined.

  There are multiple approaches to deal with the problem in the telecommunication domain. In the mentioned workshop series [7] a platform is provided for exchanging solutions in practice and theorie.

- Medicine and human body

  Feature interaction is well-known in the context of health. For medicaments a common approach is to provide a standard documentation (instruction leaflet) about the ingredients, the application and it also lists the known (potential) side-effects and interactions with other medicaments or parts of the

human body. These side-effects are more or less dangerous. The lists are developed by means of experiments during the development of the medicaments and by means of experiences and observations afterwards.

An example of a positive side-effect is a medicament called aspirin developed to be used against headache. Experiences and research proved that this medicament affects the blood-picture in a way that it lowers the danger of a cardiac infarction.

- Elevator configuration as described in [6]

  In [6] a system called VT is described which configures elevator systems at the Westinghouse Elevator Company. An elevator has cables which have some weight. This weight influences the traction ratio needed to move the car. The traction ratio influences the cable equipment (also the cable weight). Therefore, we have a circular feature dependency. In case we would like to improve the security standards and therefore increase the cable quality (which results in a higher cable weight) we have an interaction with the traction ratio which might be unforeseen if this dependency is not specified and documented. VT uses artificial intelligence (propose-and-refine approach) to deal with this problem. Individual design parameters and their inferences are represented as nodes in a network.

4 Summary and Conclusion

Feature interaction is an issue in different domains (not limited to computer science even). Due to the complexity it's usually impossible to foresee all potential interactions of the different features within one system. However, it is possible to approach the problem by identifying as many as possible unforeseen (and maybe undesirable) interactions.

Software features may be modeled by means of a suitable notation similar as designs may be modeled using UML. Notation may be found in the domain engineering discipline [1]. Even the interactions or dependencies, respectively, of features may be modeled. Incompatible combinations or default combinations may be defined already during the domain analysis phase.

In this paper we approached the terminology and proposed a classification scheme to compare different feature interaction types.

Starting with this workshop we aim at building a catalogue of problems and potential solutions. Although it is not expected that there will be one best, exact and general solution (approximative) solutions may exist for certain problem domains or specific systems. Research results in the telecommunication domain are expected to be helpful also for systems built from components.

From this catalogue we aim at an improved classification allowing to compare different interaction types or problems and a comparative catalogue of solutions. We aim at unifying problems and solution approaches.

References

[1] K. Czarnecki and U.W. Eisenecker. *Generative Programming - Methods, Tools, and Applications*. Addison-Wesley, 2000.

[2] ESPRIT Working Group 23531, `http://www.dcs.ed.ac.uk/home/stg/fireworks/workshop.html`. *FIREworks, Workshop on Language Constructs for Describing Features.*, 2001.

[3] H. Klaeren, E. Pulvermüller, A. Rashid, and A. Speck. Aspect Composition applying the Design by Contract Principle. In *Proceedings of the GCSE'00, Second International Symposium on Generative and Component-Based Software Engineering*, LNCS, Erfurt, Germany, September 2000. Springer.

[4] Merriam-Webster OnLine, http://www.m-w.com/. *Merriam Webster's Collegiate Dictionary*, 2001.

[5] E. Pulvermüller, H. Klaeren, and A. Speck. Aspects in Distributed Environments. In K. Czarnecki and U. W. Eisenecker, editors, *Proceedings of the GCSE'99, First International Symposium on Generative and Component-Based Software Engineering*, LNCS 1799, Erfurt, Germany, September 2000. Springer.

[6] M. Stefik. *Introduction to Knowledge Systems*. Morgan Kaufmann Publishers Inc., 1995.

[7] University of Glasgow, http://www.cs.stir.ac.uk/ mko/fiw00/. *Feature Interaction Workshop*, 2001.

[8] University of Glasgow, http://www.dcs.gla.ac.uk/research/hfig/features.html. *The Feature List*, 2001.

[9] University of Strathclyde, http://www.comms.eee.strath.ac.uk/~fi/. *Feature Interaction Group*, 2000.

[10] University of Waterloo, http://se.uwaterloo.ca/~s4siddiq/fi/fip.html. *Feature Interaction Problem*, 2001.