Feature Interaction Analysis: A Maintenance Perspective

Maryam Shiri
Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada
1-(514)- 848-2424 Ext. 7148
ma_shiri@cse.concordia.ca

Jameleddine Hassine
Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada
1-(514)- 848-2424 Ext. 7148
j_hassin@cse.concordia.ca

Juergen Rilling
Department of Computer Science and Software Engineering, ConcordiaUniversity, Montreal, Canada
1-(514)- 848-2424 Ext. 3016
j_rilling@cse.concordia.ca

ABSTRACT
Software systems have become more complex, with myriad features and multiple functionalities. A major challenge in developing and maintaining such complex software is to identify potential conflicts among its features. Feature interaction analysis becomes progressively more difficult as software's feature combinations and available scenarios increase. Software maintainers need to identify and analyze conflicts that can arise from feature modification requests. Our approach combines Use Case Maps with Formal Concept Analysis to assist maintainers in identifying feature modification impacts at the requirements level, without the need to examine the source code. We demonstrate the applicability of this approach using a telecommunication case study.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirement and specification – languages, methodologies

General Terms: Management

Keywords: Change Impact Analysis, Feature Interaction, Formal Concept Analysis, Software Evolution, Use Case Maps

1. INTRODUCTION
While developing software systems, it is rare that an initial system design will correspond completely with the final design or implementation of a system. Ever changing customer needs lead to requirements modifications [3]. Modifying system features can result in behavior changes when certain feature combinations occur – also referred to as Feature Interaction (FI) [11]. Modification request analysis is performed at the requirements or design level, without considering implementation details. The major goal of modification request analysis is to identify system features, their interactions and determine if certain features are potentially affected by a change request. Such feature interactions may be intentional (by design) or unexpected as a result of requirements or design decisions made without considering overall system architectures. Another source of FI occurs during maintenance when maintainers often lack an overall understanding of a system and its features prior to performing a maintenance request.

ASE'07, November 5-9, 2007, Atlanta, Georgia, USA.

The majority of existing work in FI analysis has been performed on telecommunication applications [5], due to the complexity of telephony features. As part of the specification, features are often expressed in a natural language (e.g. English) that is less precise and ambiguous. Consequently, a notation was needed to help designers describe, understand, and analyze system features in a less ambiguous way. Several notations have been introduced to describe features, and a comprehensive review of these notations can be found in [5], including Chisel Diagrams, Finite State Machines, Use Case Maps (UCM). In our work, we focus on UCMs as an example for a feature description language.

Detecting all possible feature interactions in a system is often an inherently difficult and expensive task due to the potentially large number of different feature combinations and scenarios executing these features [13]. Feature interaction analysis can be applied to identify these scenarios and features that are either affected or prone to be affected by a feature modification request [11].

In this research, we present a novel approach for FI analysis that uses Use Case Maps (UCM) [17] for the description of features and Formal Concept Analysis (FCA) [6] for grouping and filtering feature interactions. Our approach is motivated by the need to assist managers and maintainers in determining the potential impact associated with a feature modification request at the requirements level, prior to performing the actual modification and without the need for comprehending and analyzing the source code.

2. RELATED BACKGROUND
Formal Concept Analysis. FCA [6] is a mathematical approach that dates back to Birkoff in 1940 [2]. FCA is commonly used to represent and analyze information by performing logical grouping of objects with common attributes. An FCA *context* is a triple $C=(O,A,R)$ where O represents a set of objects and A, a set of attributes, with $R \subseteq O \times A$ being a relation among them [14].

FCA has gained popularity in recent years (e.g. [16]), due to (1) its programming language and application domain independence that allows users to easily define different views (using different object, attribute combinations); (2) its availability of tool support to automatically generate context tables and lattices; (3) the fact that it is a relatively inexpensive analysis, in particular compared to other more traditional dynamic dependency and trace analysis techniques. However, it has to be noted that FCA does not consider semantic information during the analysis step, limiting its applicability for certain analysis tasks. Also the flexibility in selecting different attribute-object pairs can result in difficulties selecting an appropriate context for a particular analysis task.

Use Case Maps [17] are a modeling technique that is part of a new User Requirements Notation (URN) proposal to the ITU-T [9]. UCMs capture functional requirements in terms of casual

scenarios and can represent behavioral aspects at a higher level of abstraction than for example UML diagrams. They are not necessarily bound to a specific underlying structure or implementation. UCMs are intended to complement existing UML models and can visualize an entire system at the specification level to provide a better understanding of the evolving behavior of complex and dynamic systems, such as reactive and distributed systems. Also, they can provide stakeholders with guidance and reasoning about system-wide functionalities and their behavior. UCM was originally introduced to model the behavior of telecommunication systems; however, its application domain has been extended to other domains (e.g. web based applications). For a more detailed coverage of the UCM notation, we refer the reader to [1] and [17].

UCM Example Call Model. In the following example we model a POTS telephony service using UCM, based on a 4-phase service decomposition. The four service groups provided by the telephony system are: (1) Service Request, (2) Information Check, (3) Provide the service, and (4) Disconnection.

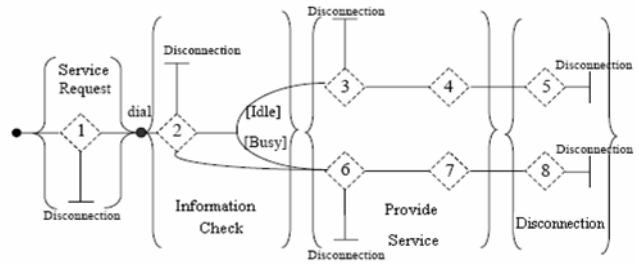

Figure 1. UCM call model (root map)

Figure 1 shows the corresponding UCM root map, with stub 1 being the call request, stub 2 being the call checking phase, stub 3, 4, 6 and 7 represent the call setup phase and stubs 5 and 8 correspond to call disconnection. The basic call model (BCM) describes the core activities involved in establishing a communication between two users A and B. The basic call can be represented through the stubs {1:Default, 2: Default,3:Default, 4: Default, 5:Default, 6: Default,7:Default, 8: Default}

Table 1. Feature interaction analysis

INTL (IN Teen Line)	Restricts outgoing calls based on the time of day
TCS(Terminating Call Screening).	Redirects incoming calls that appear on a screening list to a message
CND(Calling Number Delivery).	Allows a called telephone to receive a callee's Directory Number (DN) and the date and time
INFB (IN Free Phone Billing)	Allows subscribers to pay for incoming calls
OCS(Origination Call Screening).	Allows a call to phone numbers on a screening list
CFBL(Call Forwarding Busy Line).	Enables that all calls to the subscribing line be redirected to a predetermined number when the line is busy
VM (Voice Mail)	Allows all calls to the subscribing line are redirected to a voice mail system

In UCM, a combination of a root map, stubs, and plug-ins can be used to represent system features. The plug-in maps are sub-maps that describe locally how a feature modifies the basic behavior. Adding features to such UCM collections is often achieved by creating new plug-ins for the existing stubs or by adding new stubs containing either new plug-ins or instances of existing plug-ins. Therefore, adding features will extend the scenarios in the basic call mode by using stub and plug-ins and adding corresponding global variables (feature global variables) to the system. As a result, a scenario will only execute a feature if the feature specific plug-in(s) and enabling conditions are part of the scenario. Table 1 shows a list of features added to the original BCM, with each of these features corresponding to a submap (plug-in) and by either adding a new stub or substituting an existing stub.

Feature Interaction. Maintainers often deal with situations where one feature modifies or subverts the desired operation of another feature, or when a system functions have an incorrect behavior due to the presence of other features [11]. Identifying those scenarios and features that are either affected or prone to a modification request can be a challenge for large, feature rich systems. We define feature interaction informally as:

- One feature modifies or subverts the desired operation of another feature or
- A System feature operates incorrectly due to the presence of other features.

FI has been mainly applied to telecommunication applications and web based systems. An example for such a feature interaction in a telecommunication system is when a callee is subscribed to both VM (voice mail) and CFBL (call forwarding on busy line), and the line is busy. Suppose that caller A dials B, who is subscribed to both VM and CFBL busy. In the case that B is busy, the system cannot determine whether CFBL or VM should be activated. As a result we have in this case a non-deterministic FI occur.

Change Impact Analysis. Impact analysis focuses on identifying parts of a system that are (potentially) affected by a modification request. Change impact analysis is also the basis for regression testing, allowing for the identification of those test cases that have to be re-tested after a modification is performed. Currently, most of the research on change impact analysis focuses on source code [12] and design level analysis [4] using either traceability or dependency analysis [3]. Common to most source code base approaches is that they are computationally expensive and their applicability is limited by the supported programming language.

3. FEATURE MODIFICATION ANALYSIS

We present a novel, semi-automatic methodology for detecting potential feature interactions to support feature modification impact analysis. Our approach is based on analysis of end-to-end UCM system scenarios using FCA (Figure 2).

3.1 Generation of UCM System Level Traces

In UCM, both its abstract syntax and static semantics are informally defined as XML document type definitions. Given the absence of a formal semantic, the interpretation of UCM specifications are also left completely to the user. Furthermore, this informal representation, results in a notation that does not support symbolic execution of these scenarios. In [8], we introduced an operational semantics for UCM based on Multi-Agent ASM. The definition of the ASM formal semantics consists of associating each UCM construct with an ASM rule to model its behavior. The resulting ASM semantics are embedded in an ASM-UCM simulation engine designed for simulating and executing UCM specifications and written in AsmL [15], a high level executable specification language [8].

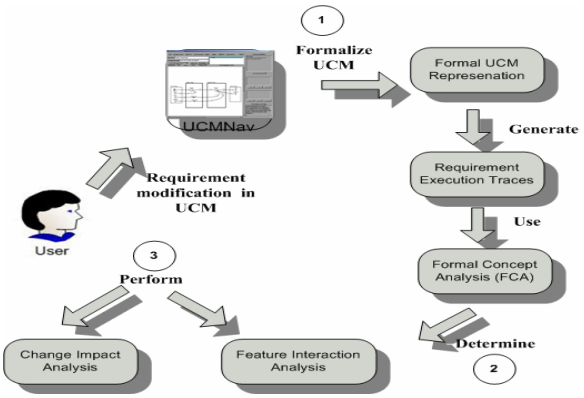

Figure 2. Process of Implementing UCM_FCA Technique

3.2 Feature Interaction Analysis

From a maintainer's perspective, one important challenge is to identify features and whether a certain feature interacts with other features. Scenarios are behavioral definitions of use cases, which typically correspond to user requirements. Formalizing UCMs allows for the execution of scenarios and the generation of traces to be used as input to the FCA. The precision of the FCA depends on the coverage achieved by these traces. Therefore, we assume UCM scenario coverage that is every scenario at the UCM level was executed at least once. For the FI analysis, we adopt a three step approach to analyze and classify features based on their interactions. The following FI classification [13] was adopted:

- *FI never occurs:* Scenarios that contain no or only one feature.
- *FI occurs*: If there exist two global feature variables (sub_F1 and sub_F2), but only one corresponding feature plug-in (plug-in_F1 OR plug-in_F2), the system has to select one of the two possible features (non-determinism).
- *FI can occur (FI prone):* A scenario is *FI prone* if there exists two feature global variables (sub-F1 AND sub_F2) and their corresponding plug-ins (plug-in-F1 and plug_in_F2). FI can occur since both feature global variables are shared by the two plug-ins.

We assume a scenario *Sc1* is a subset of all UCM scenarios SC, where $Sc1 \subset \{SC\}$. We further assume that feature scenarios will contain at least one feature implemented through one or more non-default plug-in(s). For illustration purposes, we also assume that plug-ins are annotated with the name of a feature.

In the first step of our methodology, execution traces are generated for all scenarios defined in the UCM (the collected traces form the input to the FCA analysis). In the second step, an FCA context table is generated using the following context: Global variables of features correspond to attributes and scenarios become the objects. Based on the FCA context table, one can now eliminate those scenarios which contain no or only one feature. Eliminating most of the "FI never occurs" scenarios reduces the number of scenarios that have to be further analyzed for FI in the next step. In the third step, these remaining scenarios are classified as either FI occur or FI prone, by passing down all attributes from the upper lattice levels that are associated with this scenario.

3.3 Combining UCM with FCA

The contextual representation created by FCA directly supports the dependency analysis introduced in the previous sections. The formalization of UCM allows us to execute and generate traces to be used in FCA. FCA depends on the quality and coverage achieved by the traces used for the analysis. For that reason, we assume UCM scenario coverage. That is, every scenario at the UCM level was executed at least once. Depending on the dependency type, one can create now the corresponding context by selecting the appropriate object and attribute pair. The resulting FCA context table can then be visualized as a context lattice.

The concept lattice allows for impact analysis of a requirement change, by selecting a specific feature and then traversing down the lattice until one reaches all the objects (scenarios) which share the particular attribute (feature). Without further analysis, this view would provide a too conservative estimation of the potential impacts, since there is no distinction made among different types of feature interaction. Using our approach, maintainers can now filter and detect different types of feature interactions within these scenarios to allow for a more precise analysis of potential impacts of a feature modification at requirement level.

4. CASE STUDY

In what follows, we revisit the telecommunication case study (Figure 2) to demonstrate the applicability of our methodology in guiding maintainers during feature interaction analysis and feature modification impact analysis at the requirements level. For our evaluation, we use a combination of seven features originally introduced in the feature interaction detection contests [7], [10]. Each of these features is described by an end-to-end point view and their actions are not bound to network entities.

4.1 Feature Interaction

In this section, we support maintainers during the modification analysis. The methodology starts with the scenario simplification, such as automatically removing all default plug-ins from the lattice, and eliminating those scenarios which contain no features or only one feature. In the third analysis step, the actual feature interaction analysis will take place. The analysis is based on the reduced concept lattice created by the first two analysis steps.

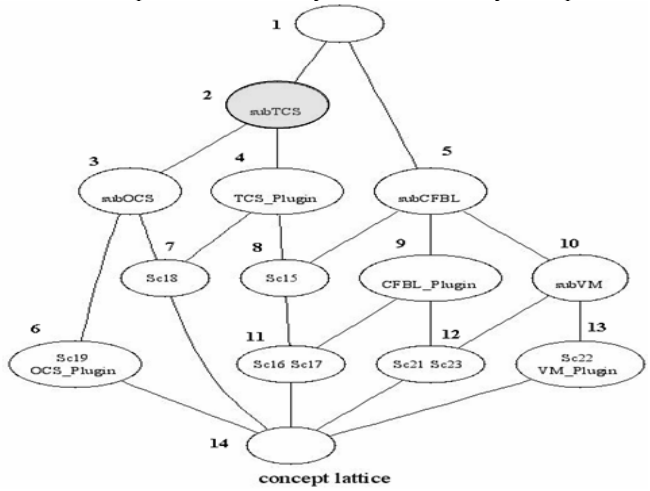

Figure 3. Feature Interaction Concept Lattice

Figure 3, shows the feature interaction concept lattice, with the remaining scenarios being objects and feature related global variables and plug-ins being the attributes in the lattice. In order to perform feature interaction analysis, a scenario is selected, for example *Sc19* (concept #6), and all its attributes will be automati-

cally passed down {*subTCS, subOCS*, and *OCS_plug-in*}. Based on the global variables, the system maintainer can now easily identify that there exist two subscription global variables subTCS and subOCS, but only one corresponding plug-in (OCS_plug-in). The fact that the *TCS_plug-in* is missing is caused by non-determinism that exist between the *TCS* and *OCS* feature. In our approach we resolve this non-determinism by executing two scenarios, with each scenario executing one possible execution path (*Sc*18 and *Sc*19). An example of a non-deterministic (*FI-occur)* feature interaction exists between the *TCS* and the *OCS* feature.

In another feature analysis example, *Sc16* (concept#11) is analyzed. In this case, again, all attributes are passed down {*subTCS, TCS_plug-in*, *subCFBL,* and *CFBL_plug-in*}. From the analysis one can see that two features *TCS* and *CFBL* and their associate plug-ins are part of *Sc16*. Based on our FI definitions, scenario *Sc16* therefore is FI prone. The analysis results for our telecommunication case study can be summarized as follows. From the 8 remaining scenarios, only two scenarios *Sc16* and *Sc17* are FI prone in the telecommunication system. The remaining 6 scenarios {Sc19, Sc18, Sc15, Sc21, Sc22, Sc23} are all *FI occur*. The remaining 15 scenarios, eliminated during the initial two analysis steps are all *FI never occurs***.**

4.2 Change Impact Analysis

As a part of the modification analysis, there is a need to identify the potential impact of a feature modification request on the remaining system. In this example, we apply our methodology to support maintainers during impact analysis. We reuse the FCA presented in Figure 3, and analyze a modification request for the feature global variable, *subTCS* (concept #2). After selecting the concept node, one can now simply pass all objects from the bottom levels up to this node, to identify the scenarios that are potentially affected by the modification request. As a result, the scenarios *Sc15*, *Sc16, Sc17, Sc18*, and *Sc19* can be identified as potentially affected by the *subTCS* modification request. Therefore, these scenarios will have to be further analyzed for potential impact and retested after the modification is performed.

5. RELATED WORK and DISCUSSION

There exists a wide range of notations and techniques in the literature to describe features [5]. Common to all of these notations is that they still require maintainers to identify and understand the interaction among features in a system. In [5], the authors provide a comprehensive review of various feature interaction analysis approaches. Formal methods based approaches are very accurate (if used correctly), but can be very costly and time consuming. FSM based approaches have the advantage of detecting all FIs, however as the number of features grows also the number of states and therefore complexity grows exponentially [13].

In [13] a two-phase UCM FI filtering method based on a SC-matrix is presented. In comparison to [13], our approach is more conservative in identifying FI prone scenarios. We are also not able to detect all FI free cases within those FI prone scenarios. Common to both approaches is that they have a similar computational complexity and both approaches can be semi-automated. Our approach differs from [13] by utilizing FCA, a more flexible, domain independent analysis technique, which is not limited to UCMs and its notation. Furthermore, the use of executable scenarios allows us to resolve some of the non-determinism during the analysis, and our approach allows for early impact analysis of feature modification request during the maintenance phase. It has to be noted that our modification analysis approach has limitations similar to other impact analysis approaches by supporting only modifications or deletion requests.

6. CONCLUSIONS and FUTURE WORK

One of the challenges during the maintenance of complex, features rich systems is the need to identify feature interactions at the system level. In this research, we present a novel approach that combines UCM with FCA to support maintainers during feature modification and feature interaction analysis at the requirements level. The analysis is performed at the requirements level without the need to have source code available. In our approach, executable end-to-end UCM system scenarios are recorded and further analyzed using FCA. As part of our future work, we plan to further validate our approach through additional case studies.

7. REFERENCES

[1] Amyot, D. Use Case Maps as a Feature Description Notation. *FIREwork Feature Constructs Workshop,* 2000.

[2] Birkhoff, G. *Lattice theory,* Rhode Island: Amer. Math.Soc, 1967.

[3] Bohner, S. A., and Arnold, R. S. An Introduction to Software Change Impact Analysis. In *Software Change Impact Analysis,* pp. 1–26, 1996.

[4] Briand, L., Labiche, Y., O'Sullivan, L. and Sówka, M. Automated impact analysis of UML models. *The Journal of Systems & Software*, vol. 79, pp. 339-352, 2006.

[5] Calder, M., Kolberg, M., Magill, E. H. and Reiff-Marganiec,S. Feature interaction: a critical review and considered forecast. Computer Networks, vol. 41, pp. 115-141, 2003.

[6] Ganter, B., and Wille, R. Formal Concept Analysis: Mathematical Foundations. *Springer*, New York, Inc. Secaucus, NJ, USA, 1997.

[7] Griffeth,N., Blumenthal,R., Gregoire,J. , and Ohta,T. Feature interaction detection contest. *Feature Interactions in Telecommunications Systems V,* pp. 327–359.

[8] Hassine, J., Rilling, J. , and Dssouli, R. Abstract operational semantics for use case maps. *Lecture Notes in CS,* pp. 366-380.

[9] ITU-T, URN Focus Group (2003), Draft Rec. Z.152 - UCM: Use Case Map Notation (UCM)., Sept. 2003.

[10] Kolberg, M., Magill, E. H. , Marples, D. and Reiff, S. Second feature interaction contest. *Workshop on FI in Telecommunication Networks and Distributed Systems,* pp. 293-310, 2000.

[11] La Porta, T. F., Lee, D., Lin, Y. J. and Yannakakis, M.,"Protocol Feature Interactions", *Int. Conf. on Formal Description Tech. for Distributed Systems and Communication Protocols,* pp.59-74, 1998.

[12] Law, J. and Rothermel, G. Whole program path-based dynamic impact analysis. In *proceedings of the International Conference on Software Engineering,* pp. 308–318, 2003.

[13] Leelaprute, P., Nakamura, N. , Matsumoto, K. and Kikuno, T. Design and Evaluation of Feature Interaction Filtering with Use Case Maps. *NECTEC Technical Journal*, pp. 581-597, 2005.

[14] Lindig, C. Concept-based component retrieval. *Work. on Formal Approaches to the Reuse of Plans, Proofs, and Programs,* 1995.

[15] Microsoft, "AsmL for Microsoft.Net," http://research.microsoft.com/foundations/asml, April/2007.

[16] Tallam, S., and Gupta, N. A concept analysis inspired greedy algorithm for test suite minimization. *Program Analysis for Software Tools and Eng*. pp. 35-42, 2005.

[17] UCM, «Use Case Maps Web Page and UCM Users Group», http://jucmnav.softwareengineering.ca/twiki/bin/view/Main/WebHome.