Toward Automated Feature Model Generation from UML Use Case and Class Diagrams

Tasneem Yousef* and Said Ghoul*
Philadelphia University, Amman, Jordan

Abstract

The Feature Model is one of the most important concepts in the Software Product Line development process, as it helps to represent commonalities and variabilities between software products. Several research works are currently conducted to generate feature model of a software product line from its requirements document. However, the generated feature models do not include data and actor aspects which are essential parts of any software product line. The work presented in this paper proposes an approach handling this insufficiency. This approach defines a methodology toward automated feature model generation from requirement documents limited to Unified Modelling Language use case and class diagrams. The use case is specified with a use case description template enhanced with information required for feature model. The class diagram is specified in a textual form obtained by translating the diagram into java programming language. The target generated feature model is enhanced by the introduction of data and actor concepts. An evaluating real case study (quality assurance at Philadelphia University), was used to evaluate the feasibility of the proposed approach and obtained results show its practical benefits.

Key Words: Feature model (FM), software production line (SPL), unified modelling language (UML) use case, UML class diagram, feature model generation approach.

1 Introduction

Building software from scratch is complex and expensive, and there is a high probability of mistakes. But, the concept of SPL helps to analyse a set of software that share some parts and are designed for a specific domain [7]. This approach allowed reuse of commonalities and variabilities that resulted. There is a core asset that represents the commonalities and variabilities, including the variability model such as the FM [1, 4-5]. It helped to represent the variable and common characteristics of software, to make decisions, and to avoid mistakes [2]. A FM is one of the most efficient ways to represent and manage the variance obtained from products during the SPL process. It is a visual representation of features in a hierarchy, called a feature tree, at the top of what are the main features and at the bottom are the features that branch off from the main features and the relationships between them [3].

Researchers have introduced many approaches for generating FMs. These approaches define input requirements document, generation processes, and output generated FM. Researchers have used many forms of input documents. Some of them used features set & UML-class diagram as input as in ModelVars2SPL approach [2]. Others used Hasse diagram as inputs in Equivalence Class Feature Diagram approach [3]. The feature models obtained through these approaches may not contain all the relationships and features covering all the parts of an SPL process [2, 3, 4, 6, 10, 13]. In fact, the researchers have developed automated generation methodologies, but the FM generated through these methodologies does not include all the features and relationships [2], particularly data and actor features which are important in any SPL.

To solve this insufficiency in the FM generated by current methodologies, this paper proposes an approach toward automated FM generation from the requirements specification documents: A UML [9 use case description (by semi-formal language) and a UML class diagram translated into Java language. The final FM contains traditional features, additional features (actor and data), relationships between them, and constraints.

An evaluating real case study (quality assurance at Philadelphia University), was used to evaluate the feasibility of the proposed approach, and obtained results showed its practical benefits.

2 Background

2.1 Feature Models

FMs are language, used for system visual description, which serves to determine the scope of the product line by the features of its products. It represents the features that the product may or may not have, by defining the features, relationships, and constraints between them. A FM is hierarchy called feature tree that contains the main features at the top of the tree and the features extending from them. Features can be mandatory or optional. They are linked together with relationships [3]: *OR* relation (the main feature consists of one or more features extended from it), *XOR* relation (the main feature consists of only one of the features extended from it), *Exclude* relation (The two features that found this relationship between them, cannot

*Research Laboratory on Bioinspired Software Engineering. Email: tasneem.fy18@gmail.com, shgoul@philadelphia.edu.jo.

be in the same product), *Imply* relation (the selected feature requires another feature, to be configured properly). Some researchers have represented some relationships (as exclude and imply) outside the feature model. It was represented by textual relationships called constraints, to reduce the complexity of getting a network of relationships, difficult to understand and maintain in FM [11].

2.2 Software Requirements Document

*Software r*equirements document includes its elicited functional and non-functional requirements, their evaluation, formalization, and quality [12]. The elicitation allows gathering all requirements. The evaluation allows selecting the alternative requirements according to suitable criteria (priority, feasibility, security, quality, cost, etc.). The formalization allows specifying selected requirement using formal languages (Algebraic, Z, SDL, Petri-nets, etc.) in order to facilitate their automated translation into design and code. The quality allows evaluation of the obtained software requirement according to some quality norms and indicators [4].

3 An Application Case

The Quality Assurance (QA) agenda system at Philadelphia University is used as an application case to illustrate and evaluate the feasibility of the concepts presented in this paper. This system manages 16 academic weeks, but in this study the focus is on the seventh and twelfth weeks because they are the most important and significant. The QA agenda requirements are used as inputs to the proposed approach. These inputs are specified with two UML models: use case (Figure 1a) Using these inputs, an enhanced FM [14] is generated (Figure 2). Traditional aspect of that FM includes four main features student answers, student marks, control, and store checklist. All of them are mandatory features in the system. It also contains description and Class diagram (Figure 1b) translated into java. an AND relationship that extends from some of the main features, and a group of sub-features such as the "student answers" feature, that extends from it (discuss the exam with student, store a soft copy of the exam sheets, store marking schema). The new specification of that FM includes data and actor features, as well as inherent constraints.

4 Feature Model Generation Approach

4.1 Enhanced FM with Additional Features (Data and Actor)

As consequence to current relevant research works analysis [14], some enhancements to traditional FM are proposed (Figure 3). They are related to actors, data, and constraints over them.

4.2 Generation Process

The generation approach (Figure 4), is composed of three main components: the inputs that are UML models, the generation process, which goes through a set of steps for extracting the target FM model from requirements, and finally ends with the output, which is the generated FM.

4.3 The Req-To-FM Process

The *Req-to-FM* process (Figures 5a and 5b) is built on the basis of extraction from UML models (use case description and class diagram translated in java) of the target FM. It starts with two parallel processes building partial FMs through two phase extraction and building:

- *UC-to-FM*: It extracts traditional features, actors, constraints, and inherent relationships from use case description and builds up a UC-FM.
- *CD-to-FML:* It extracts data features, inherent relationships, and constraints from class diagram in java form and builds up a CD-FM

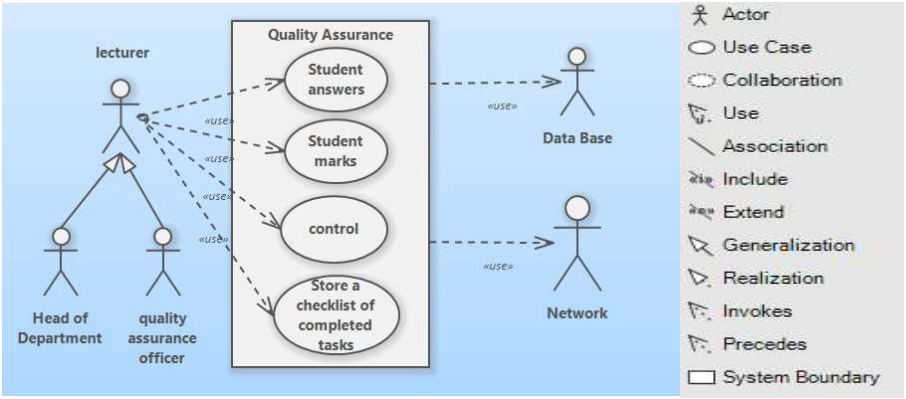

Figure 1a: QA agenda UML Use Case Diagram. This figure depicts the following:
- QA agenda actors with their relationships: QA officer, lecturer, head of department, database, and network.
- Three use cases classes: Students answers, students' marks, control and store a checklist of completed tasks

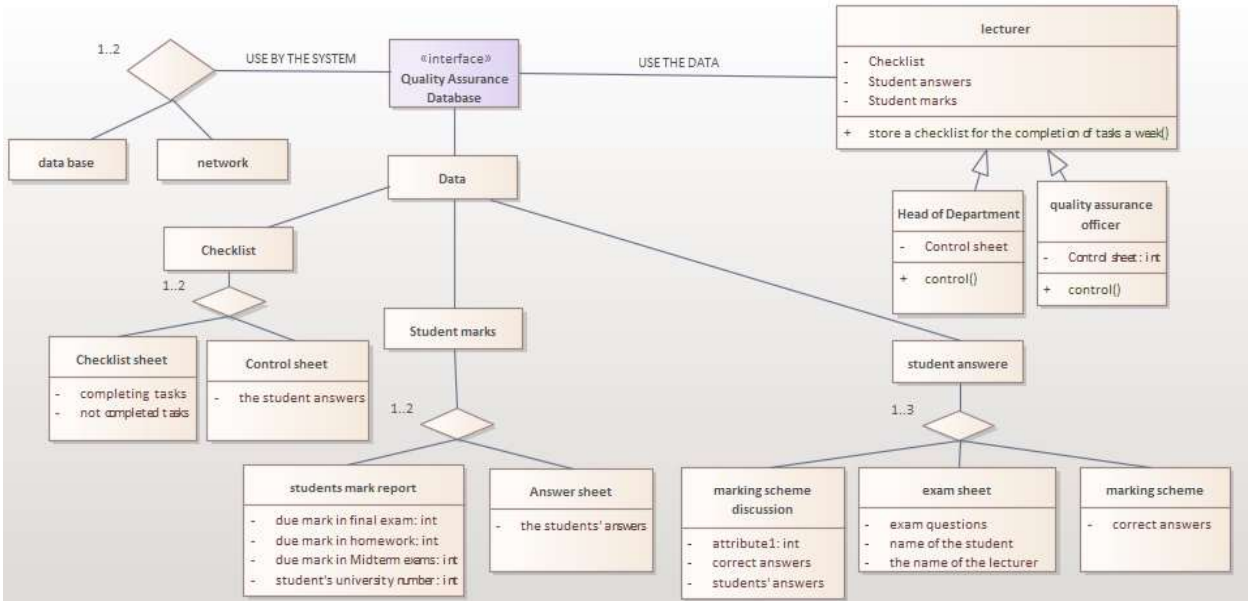

Figure 1b: QA agenda UML Use Case Diagram. This figure depicts the following:
- Actor Classes: QA officer, lecturer, head of department, database, and network.
- Data Classes: Exam sheet, answer sheet, marking scheme, marking scheme discussion, and student marks report

The process Merge-FMs ends Req-to-FM by merging the UC-FM with the CD-FM and producing the final target FM.

4.4 The UC-To-FM- Processes

This process extracts information from a UML use case description (in a logic language) and build up a UC- (FM. The extraction is carried out through four parallel processes: Features Extraction, Relation and/or/xor) Extraction, Actor Boundary Extraction, and Constraint Extraction (Figures 6a and 6b).

4.5 The CD-To-FM- Processes

This process extracts information from a UML class diagram (converted into Java language) and build up a CD-FM. The extraction is carried out through three parallel processes: Data Features Extraction, Relation and/or/xor) Extraction, and Constraints Extraction (Figures 7a and 7b).

5 Case Study

This section presents the feasibility study for the illustrative example (discussed in Sections 3) by the application of the UC-to-FM and CD-to-FM on a concrete business domain, which is the QA assurance agenda at Philadelphia University.

The Figure 8a shows partial application of the process UC-to-FM extraction phase. The outputs are the traditional part of the FM: a set of main features (student answers, student marks, control, store checklist for the completion of tasks a week), all of them are mandatory, and a set of sub-features (discuss the exam with student, store a soft copy of the exam sheets, …), and a set of relationships over them represented by the AND relationship.

The Figure 8b shows partial application of the process CD-to-FM extraction phase. The outputs are: a set of features that are divided into data and actor, and all the main features are mandatory, and a set of relations between them represented by the relationships (and, or, is-a).

After completing the components extraction step, the process moves to the rest of the process steps (depicted in Figures 5a and 5b). It ends with the FM depicted in Figure 2.

6 Comparison with Similar Works and Evaluation

Based on the current research in the field of FM generation, the following criteria are selected to evaluate the proposed approach and compare it with modern relevant studies.

(1) Improving the FM concepts to cover major aspects of SPL (data, actors, …).
(2) Using UML like -Diagrams (UC, CD, ...) as inputs to the generation approach.
(3) Formal definition of the FM generation approach.
(4) Completely automated generation approach.

In the study [2], the authors used a method to generate FMs called ModelVars 2SPL, it generates FMs from UML models

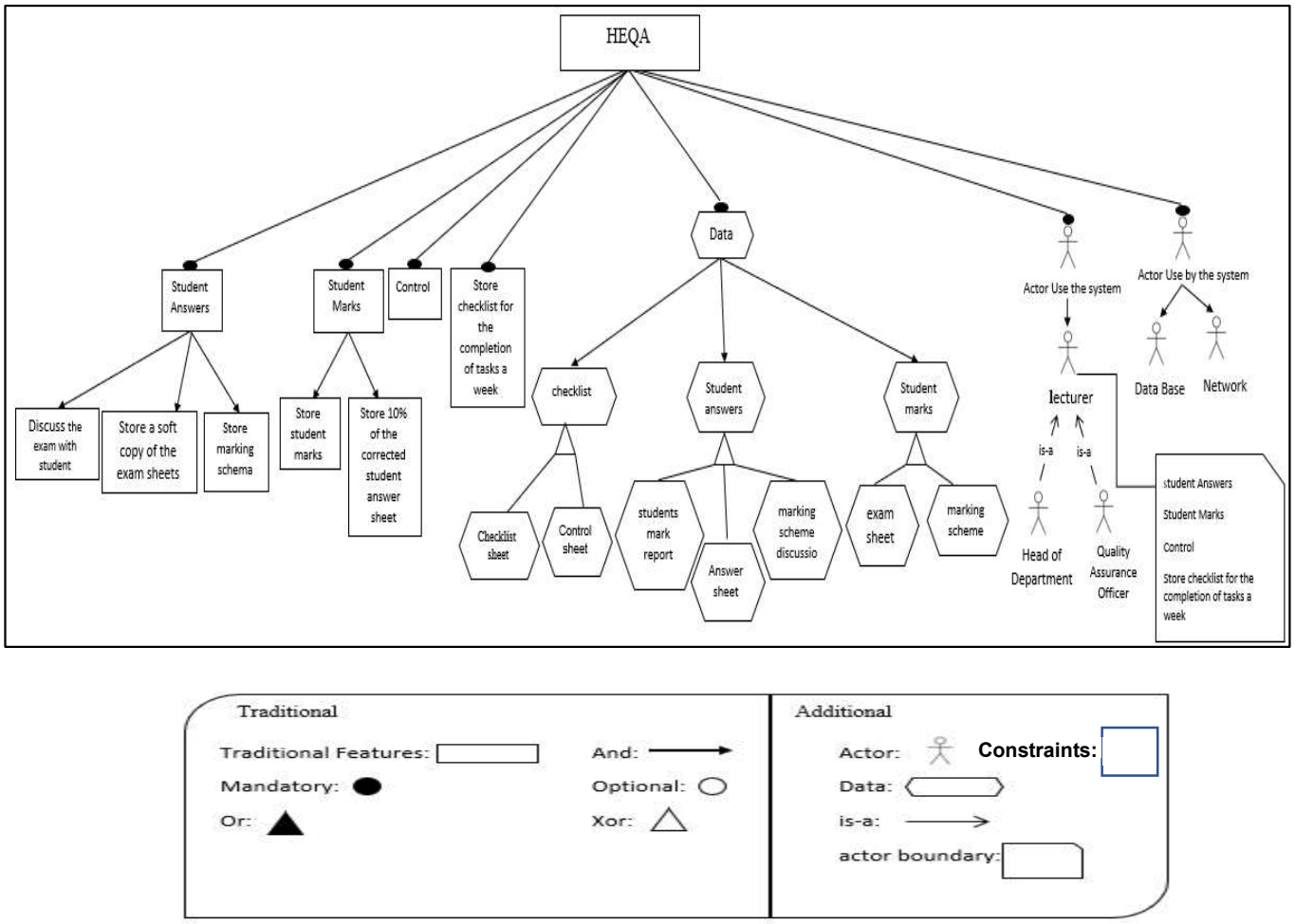

Figure 2: Generated FM from use case (Figure 1a) and class diagram (Figure 1b) for QA agenda at Philadelphia University. This FM was enhanced with additional features (data and actor), their relationships, and their inherent constraints

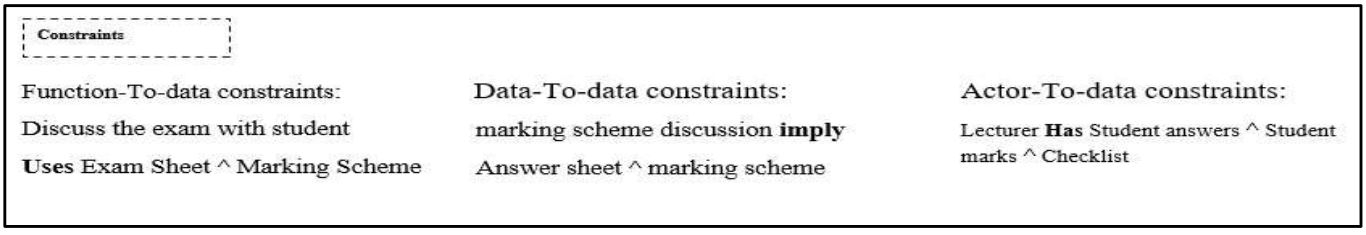

Figure 3: Generated FM uses traditional and additional notations. Additional notations specify actors, data, and constraints over them [14]. These constraints concern relationship between (function, data), (function, actor), (data, data), and (actor, data)

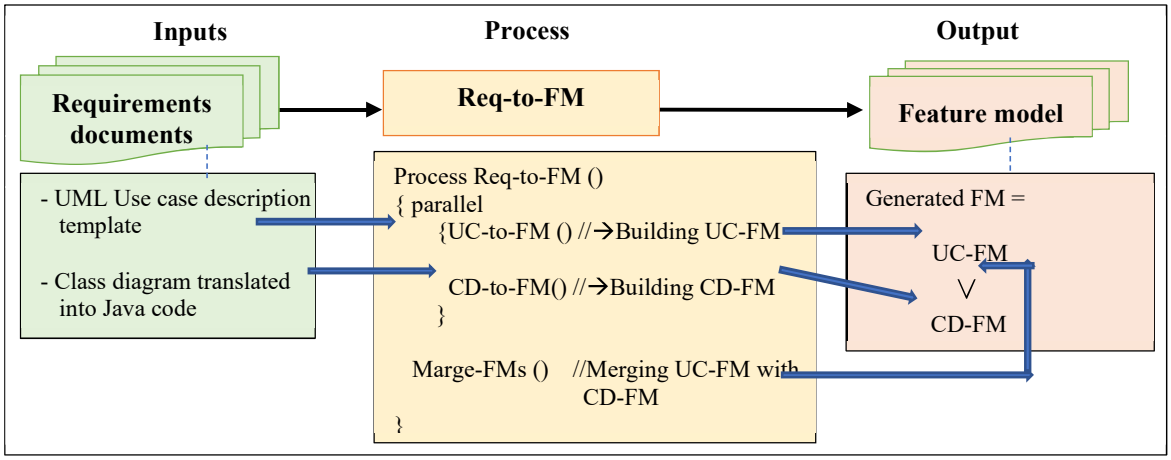

Figure 4: FM generation approach: The Req-to-FM process generates FM from UML Use and Class diagrams.

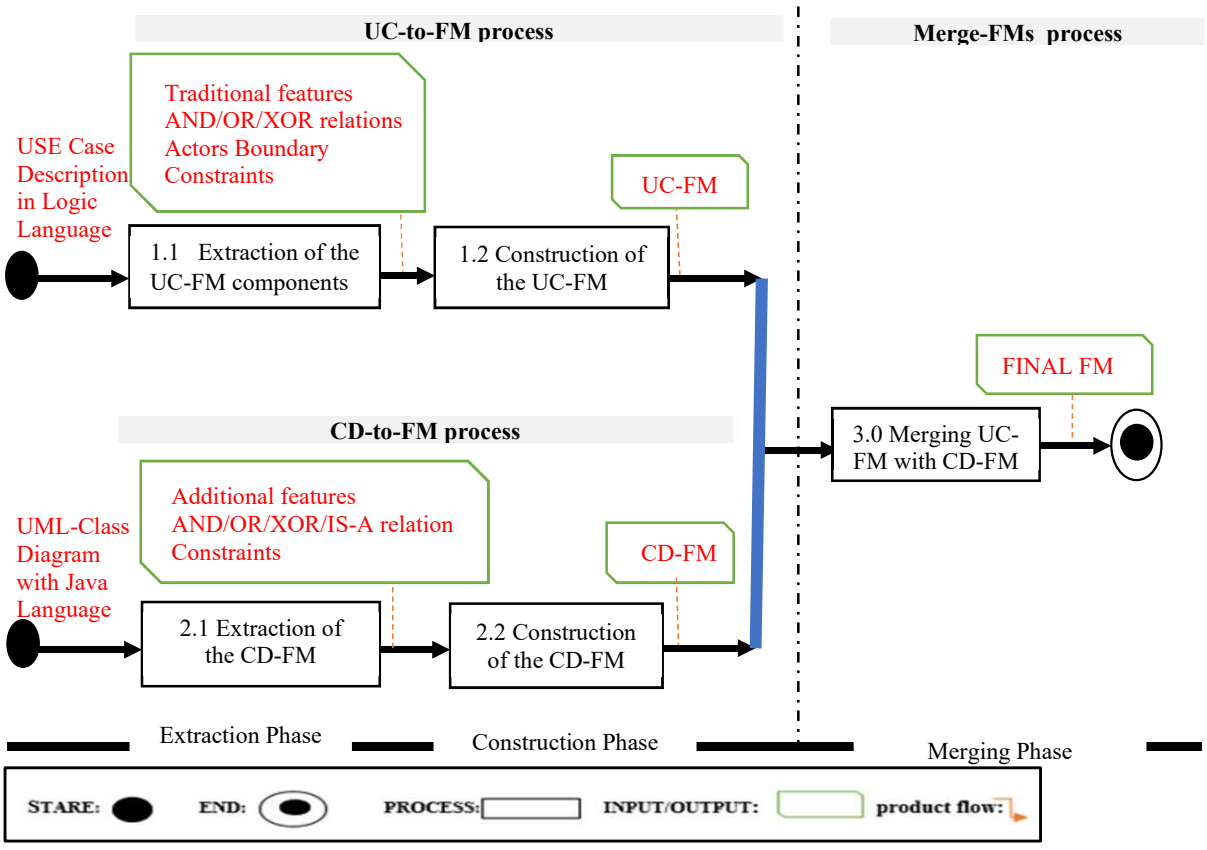

Figure 5a: The Req-to-FM process using UML notations

```
Include UC-to-FM (in UML Ucd, out FM Fm CD-to-FM (in Java Cdjava, out FM Fmcd);
Process Req-to-FM (in UML UC-Description, CD-Java; out FM TargFM)
Begin
   Parallel:
    {
  UC-to-FM (UC-Description, UC-FM);
      CD-to-FM (CD-Java, CD-FM);
    }
   TargFM ← Merge-FMs (UC-FM, CD-FM)
End Req-to-FM
```

Figure 5b: The Req-to-FM process using algorithmic notations

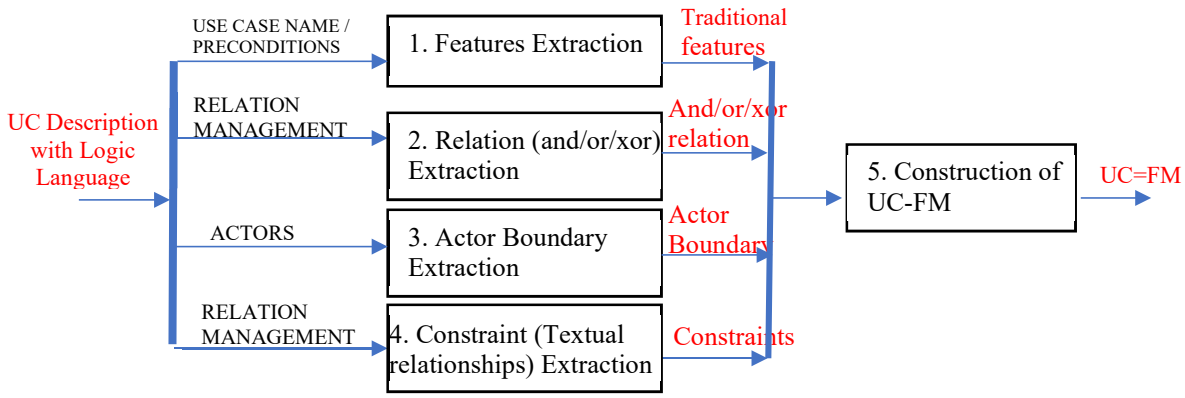

Figure 6a: The UC-to-FM process using UML notations

```
Include Features-Extraction (in UC UC-Description, out Feature Traditional-Features) Relation-and/or/xor-Extraction (in UC UC-Description, out Relation Relations) Actor-Boundary-Extraction (in UC UC-Description, out Actor Actor-Boundary) Constraints-Extraction (in UC UC-Description, out Constraint Constraints)
Process UC-to-FM (in UML UC-Description, out FM UC-FM);
  Feature Traditional-Feature; Relation Relations, Actor Actor-Boundary;
  Constraint Constraints
Begin
   Parallel:
    {
      Features-Extraction (UC-Description, Traditional-Features)
      Relation-and/or/xor-Extraction (UC-Description, Relations)
      Actor-Boundary-Extraction (UC-Description, Actor-Boundary)
      Constraints-Extraction (UC-Description, Constraints)
    }
UC-FM ← Construct-FM (Traditional-Feature; Relations, Actor-Boundary, Constraints)
End UC-to-FM
```

Figure 6b: The UC-to-FM process using algorithmic notations

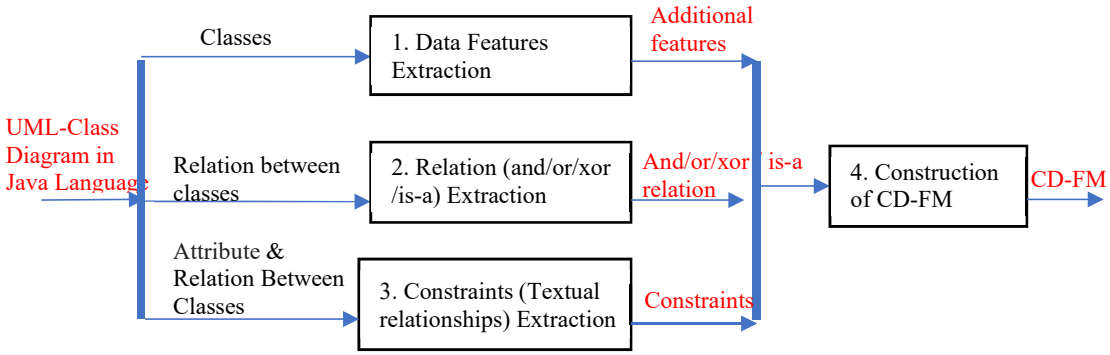

Figure 7a: The CD-to-FM process using UML notations

```
Include   Data-Features-Extraction (Java CD-Java, feature Additional-features)
          Relation-and/or/xor /is-a-Extraction (Java CD-Java, relation Additional-relationships)
          Constraints-Extraction (Java CD-Java, Constraint Additional-constraints)

Process CD-to-FM (in Java CD, out FM CD-FM);
  Feature Additional-Features; Relation Additional-relationships, Constraint Additional-constraints
Begin
   Parallel:
    {
     Data-Features-Extraction (CD-Java, Additional-features);
     Relation-and/or/xor /is-a-Extraction (CD-Java, Additional-relationships);
     Constraints-Extraction (CD-Java, Additional-constraints);
    }
   CD-FM ← Construct-FM (Additional-features, Additional-relationships, Additional-constraints);
End CD-to-FM
```

Figure 7b: The CD-to-FM process using algorithmic notations

and a set of enumerated features. New features may be discovered during the generation of the FM, and features that may encounter a problem during the generation process might be identified. But this technique does not detect all feature groups. Sometimes not disclosing all features can cause problems during the development stage. The aim of generating feature models is to identify all possible features for developing a product.

In another study [3], the work was done to improve relations from what was used as input. Hasse was a diagram used to describe the concept of lattice associated with the formal context. The equivalence class feature diagram technique was used to generate FMs. But this technique does not detect all feature groups.

In [4] the authors proposed a manual generation of FMs. This process was divided into several stages, including identifying features, groups of features, commonality, variability, and ending with the generation of a FM. The researchers also specified the type of input, which was in the form of an activity diagram. This method is manual and it was recommended to be automated.

The researchers, in [11], have written the CVL language, which is used to represent a FM. It contains an Imply relationship in two forms. The first one repeating the feature and linking it to the feature that brings them together. The second one represents it as constraint to avoid complexity and the occurrence of a network that makes it difficult for the reader to understand the model.

The comparison between what this study proposes and the recent works mentioned above will be summarized as it follows:

(1) Improving the FM concepts to cover major aspects of SPL (data, actors, ...)

All the previous works relied on generating the traditional aspect of the feature model only. In this paper, core feature model concepts are developed and enhanced to support more SPL aspects. In fact, data and actor features were added with

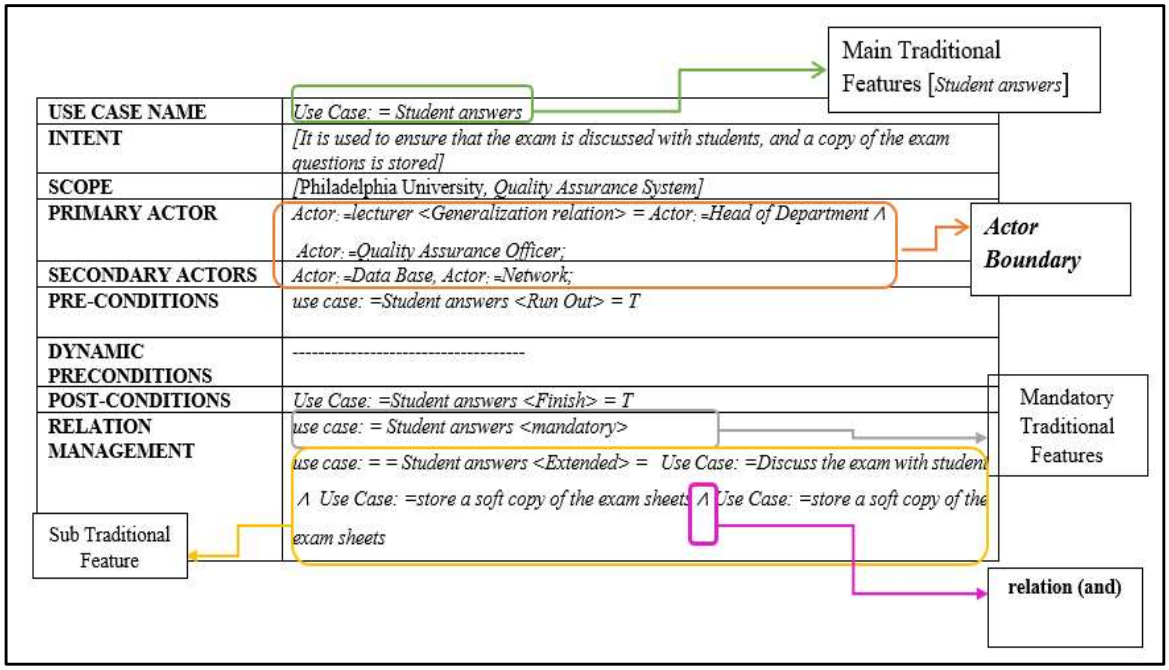

Figure 8a: An Output of UC-to-FM extraction phase, applied on QA agenda at Philadelphia University

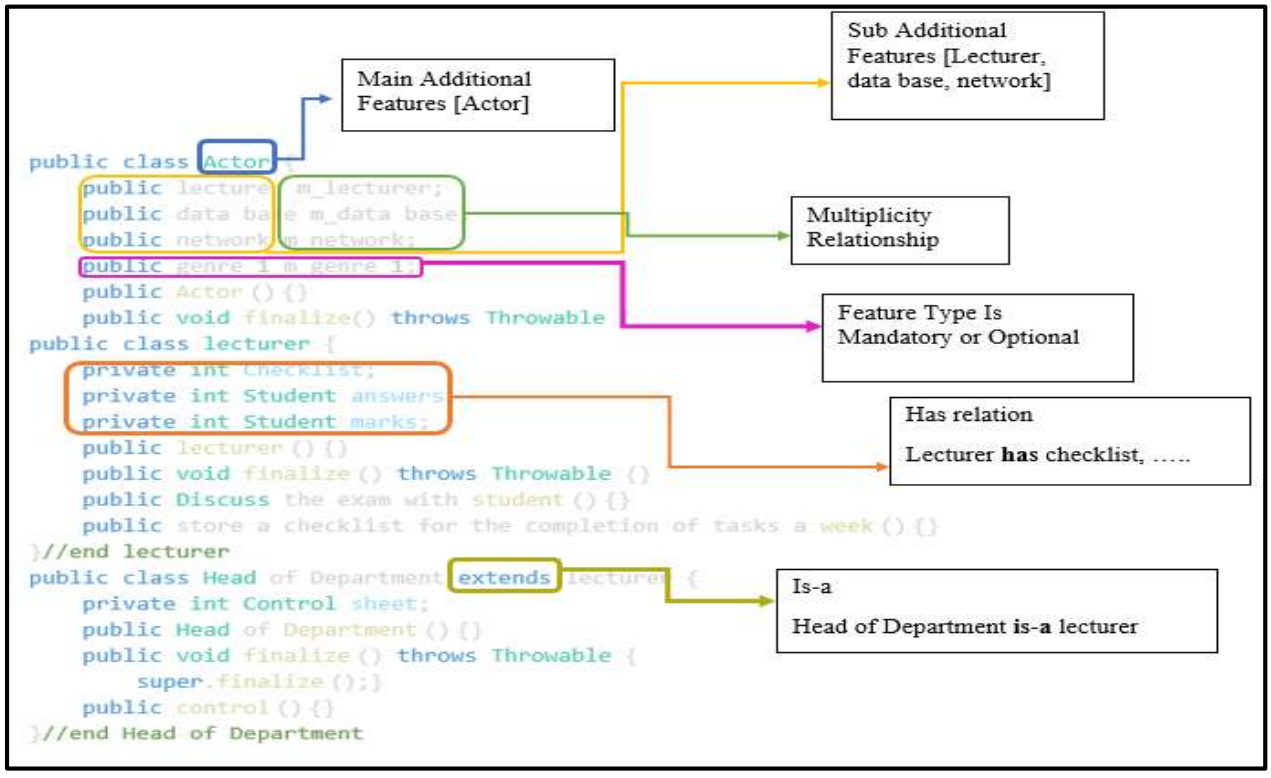

Figure 8b: An output of CD-to-FM extraction phase, applied on QA agenda at Philadelphia University

their inherent relationships.

(2) Using UML like -Diagrams (UC, CD, ...) as inputs to the generation approach

Not all of the above works used UML like diagrams to represent the requirements. In this paper, a use case description with a logic-based language was used along with class diagram translated into Java Language. The use of these diagrams reduces ambiguity and lack in requirements; as clear and understandable requirements help in the correct generation of the feature model.

(3) Formal definition of the FM generation approach

All the above works used several methods to present their work, but rare are the works that formalized their approach. In this paper, the generation approach was formalized at all levels: inputs using UML, the process using UML and algorithmic notations, and the output with an enhanced FM formalism.

(4) Completely automated generation approach

A few of the previous works above provided automatic generation for the traditional aspect of feature models, and the rest of the workers provided manual feature generation with an approach to generating. This paper presented a semi-automated approach (without a completely operational tool) for FM generation that starts from the input and ends with the target FM. The inputs and the output were formally defined using appropriated languages and tools, whereas the algorithms of the generation process were specified.

7 Conclusion

According to the previously presented study on FM generation to represent the commonalities and variabilities from SPL, it was stated that the current methods only generate traditional feature models and do not represent the whole aspects of a system. The feature model generation methods do not deal with the data and actor aspects, they only dealt with the functional aspect of a system. In this work, it is proposed that the inputs be represented by UML models. An approach has been developed consisting of two parallel processes, the first process consists of extraction of a UC-FM from use case description, and the second process consists of extraction of CD-FM from class diagrams. The approach ends with merging UC-FM with CD-FM in a target FM. The proposed approach feasibility was validated on a real business domain that is the quality assurance in higher education limited to Philadelphia University Quality Assurance Agenda system. A comparative evaluation with the closest recent works, based on some significant and usual criteria, was conducted. It worked out to clarify the value of the proposed approach regarding some relevant works. As future work, the approach could be fully automated by providing an automated supporting tool. Other input models could be investigated in order to identify the most suitable for the generation process in terms of cost and quality. The limit of the generated FM could be enlarged by rely identifying the needed and optimal information to include.

References

[1] P. Arcaini, A. Gargantini, and M. Radavelli, “Achieving Change Requirements of Feature Models by an Evolutionary Approach,” *J. Syst. Softw.*, 150: 64-76, 2019.

[2] W. K. G. Assunção, S. R. Vergilio, and R. E. Lopez-Herrejon, “Automatic Extraction of Product Line Architecture and Feature Models from UML Class Diagram Variants,” Information and Software Technology, 117:1-19, 2020.

[3] J. Carbonnel, M. Huchard, and C. Nebut, “Modelling Equivalence Classes of Feature Models with Concept Lattices to Assist their Extraction from Product Descriptions,” *J. Syst. Softw.*, 152:1-23, 2019.

[4] O. Komarudin, D. Adianto, and A. Azurat, “Modeling Requirements of Multiple Single Products to Feature Model,” *Procedia Comput. Sci.*, 161:107-114, 2019.

[5] L. Li, Y. Zheng, M. Yang, J. Leng, Z. Cheng, Y. Xiem, P. Jiang, and Y. Ma, “A Survey of Feature Modeling Methods: Historical Evolution and New Development,” *Robot. Comput. Integr. Manuf.*, 61(101851):1-16, September 2020.

[6] Z. Liu, N. Japkowicz, R. Wang, Y. Cai, D. Tang, and X. Cai, “A Statistical Pattern based Feature Extraction Method on System Call Traces for Anomaly Detection,” *Inf. Softw. Technol.*, 126:1-13, 2020.

[7] M. Marques, J. Simmonds, P. O. Rossel, and M. C. Bastarrica, “Software Product Line Evolution: A Systematic Literature Review,” *Inf. Softw. Technol.*, 105:190-208, January 2019.

[8] L. Montgomery, D. Fucci, A. Bouraffa, L. Scholz, and W. Maalej, “Empirical Research on Requirements Quality: A Systematic Mapping Study,” Requirements Engineering, 27(2):183-209, 2022.

[9] OMG, Unified Modeling Language 2.5.1, 2017

[10] D. Pantazis, P. Goodall, P. P. Conway, and A. A. West. “An Automated Feature Extraction Method with Application to Empirical Model Development from Machining Power Data,” *Mech. Syst. Signal Process.*, 124:21-35, 2019.

[11] I. Reinhartz-Berger, K. Figl, and Ø. Haugen, “Investigating Styles in Variability Modeling: Hierarchical vs. Constrained Styles,” *I nf. Softw. Technol.*, 87:81-102, 2017, doi: 10.1016/j.infsof.2017.01.012.

[12] M. K. Sabariah, P. I. Santosa, and R. A. Ferdiana, “Proposed User Requirements Document for Children’s Learning Application,”, *International Journal of Advanced Computer Science and Applications*, 11(9):317-324. 2020.

[13] A. G. Salguero and M. Espinilla, “Ontology-based Feature Generation to Improve Accuracy of Activity Recognition in Smart Environments,” *Comput. Electr. Eng.*, 68:1-13,

March 2018.
[14] T. Yousef, E. Nafar, S. Ghoul, L. Quraan. “Automatically Generated Feature Model from Requirements: Toward an Enhanced Formalism,” The 13th International Conference on Information and Communication Systems (ICICS2022), IEEE Xplorer, 2022

Ahlem Yousef holds a MSc in Software Engineering from Philadelphia University, Amman, Jordan.

She is a member of the research team in the Research Laboratory on Bio-inspired Software Engineering. Her research interest includes: Requirements specification, automated specifications generation, and requirements traceability.

Said Ghoul holds a PhD in Software Engineering and a Doctorate in Science in Software Processes.

He is Full Professor of Software Engineering at Philadelphia University, Jordan and Chair of the Research Laboratory on Bio-inspired Software Engineering.

His research interest includes: Bio-inspired Design and Modeling, Variability aspects, and Self-adaptive systems.