2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications

The UML *«include»* Relationship and the Functional Refinement of Use Cases

Sofia Azevedo, Ricardo J. Machado
Dep. de Sistemas de Informação
Universidade do Minho
Guimarães, Portugal
{sofia.azevedo,rmac}@dsi.uminho.pt

Alexandre Bragança
Dep. de Eng. Informática
ISEP
Porto, Portugal
alex@dei.isep.ipp.pt

Hugo Ribeiro
Primavera BSS
Braga, Portugal
hugo.ribeiro@primaverabss.com

***Abstract*—Developing software with model-driven approaches involves dealing with diverse modeling artifacts such as use case diagrams, component diagrams, class diagrams, activity diagrams, sequence diagrams and others. In this paper we focus on use cases for software development and we analyze them from the perspective of detail. In that context we explore the UML (Unified Modeling Language) *«include»* relationship. This work allows understanding the use case modeling activity with support for refinement and provides for specific guidelines on how to conduct such activity.**

Use case; functional refinement; functional decomposition; detail; «include»; «refine»

I. INTRODUCTION

Use case diagrams are one of the modeling artifacts that modelers have to deal with when developing software with a model-driven approach. This paper envisions use cases according to the perspective of detail (which has to do with the abstraction level use cases may be situated at and implies refinement as it will be exposed).

Use cases can be more or less detailed, which means that they can be refined. The refinement of a use case results in lower-abstraction-level use cases. The lowering of the abstraction level shall be represented in the diagrams with a new kind of relationship we will present ahead in this paper: the *«refine»* relationship. In this paper we consider that refinement is at the functional perspective. We explain why we consider that the *«include»* relationship is not adequate to support the refinement in use case diagrams. It shall be noted that in our approach use cases are still use cases, representing external functionality of the system that can be performed by the actors (a use case still represents observable value to an actor, despite being more or less detailed). Refining use cases is important to incrementally introduce user requirements in the design of the software system.

The *«refine»* relationship represents the refinement of use cases. The refinement of use cases is an approach to deal with the problem of complexity in the modeling activity. This paper's contribution is on the understanding of the use case modeling activity with support for refinement, providing specific directives on how to conduct such activity in a systematic way. We illustrate our approach with the Fraunhofer IESE's GoPhone case study [1], which presents a series of use cases for a part of a mobile phone product line particularly concerning the interaction between the user and the mobile phone software. We consider use cases in different abstraction levels according to the *«refine»* relationship. We also propose an extension to the UML (Unified Modeling Language) metamodel [2] in order to support both the concrete and abstract syntaxes of the refinement of use cases. In this paper we focus on the refinement support as well as on the process point of view with regards to the use case modeling activity.

The remainder of this paper is organized as follows: Section 2 presents the work of other authors on the refinement of use cases; Section 3 introduces the process of refining use cases from the detail perspective; Section 4 is about defining the *«refine»* relationship, discussing the difference between the *«include»* and the *«refine»* relationships and how the *«refine»* relationship is the one applicable to represent the refinement of use cases; Section 5 clarifies the process of modeling use cases when refinement is involved and gives some guidelines in order to conduct that process; Section 6 illustrates our approach to use case modeling with support for refinement with the GoPhone case study; and finally Section 7 provides for some concluding remarks.

II. RELATED WORK

Refinement has been treated over the years. Paech and Rumpe provide in [3] for a formal approach to incrementally design types through refinement. Types represent the static part of a system captured through software models, and consist of attributes and operations. Our approach is not formal and relates to the refinement of external functionalities of software systems, which shall be taken into account before the static part of those systems. Quartel*, et al.* propose in [4] an approach for action refinement consisting of replacing an abstract action with a concrete activity (composition of actions) based on the application of rules to determine the conformance of the concrete activity to the abstract action. Again our approach relates to a perspective that shall be taken into account before behavior. Darimont and van Lamsweerde talk in [5] about goal refinement. In their approach the refinement process is guided by refinement patterns used for pointing out missing elements in refinements. This time our approach to refinement relates to a perspective that shall be taken into account after goals. Schrefl and Stumptner face in [6] refinement as the decomposition of states and activites into substates and subactivities though inheritance. Our approach to refinement

978-0-7695-4170-9/10 $26.00 © 2010 IEEE
DOI 10.1109/SEAA.2010.65

considers that refinement shall not be tretaed through generalization as it will be stated further on in this paper. Mikolajczak and Wang present in [7] an approach to vertical conceptual modeling of concurrent systems through stepwise refinement using Petri net morphisms. Our approach to refinement is not formal. Batory created a model (the AHEAD model [8]) for expressing the refinement of system representations as equations. Despite his approach being based on stepwise refinement he worked at a code-oriented level. The work we are presenting in this paper allows refining (also in a stepwise manner) software models that shall be handled before code is handled during the software construction phase.

Cherfi*, et al.* [9] (in their work on quality-based use case modeling with refinement) describe the refinement process as the application of a set of decomposition and restructuring rules to the initial use case diagram. Their approach is iterative and incremental. It consists of decomposing the initial use case diagram into smaller and more cohesive ones to decrease the complexity of the diagram and increase its cohesion. In their approach a use case is a set of activities that varies according to scenarios, which are flows of actions belonging to those activities. In the first phase of the refinement process a use case is decomposed into other use cases according to one or more scenarios. The second phase of the refinement process is about eliminating the redundant activities that compose the use cases obtained from the first phase, which generates *«include»* relationships. Their approach allows defining the *«include»* relationships based on the commonality among the system's activities performed for different scenarios. Our approach considers that the *«include»* relationship is defined based on the non-stepwise textual descriptions of the use cases and that stepwise descriptions (like those considered by Cherfi*, et al.*) shall be treated separately (stepwise textual descriptions are structured textual descriptions in natural language that provide for a stepwise view of the use case as a sequence of steps, alert for the decisions that have to be made by the user and evidence the notion of use case actions temporarily dependent on each other; Cockburn presents in [10] different forms of writing textual descriptions for use cases). Also in the approach of Cherfi*, et al.* to refinement, use cases are not actually detailed (like in ours), rather they are decomposed without detail being added to the description of those use cases.

Pons and Kutsche [11] present the refinement activity as a way to trace code back to system requirements and system requirements back to business goals, which allows verifying whether the code meets the business goals and the system requirements as expected in the specification of the system. Although these authors do not formally extend the UML metamodel to incorporate a new kind of relationship between use cases, they use this new kind of relationship between diagrams. But Pons and Kutsche use the relationship to connect two use cases belonging to two different diagrams, whereas our vision is that the refinement relationship shall be established between one use case (a diagram) and two or more use cases (another diagram) to distinguish the different levels of abstraction both diagrams are situated at. Despite that Pons and Kutsche distinguish between refinement by decomposition and refinement by specialization, they achieve refinement by specialization through a generalization relationship between use cases that belong to the same diagram. Our position towards refinement is that the refinement relationship may be defined *by decomposition* but it is established between different diagrams as the use cases connected through the refinement relationship are situated at different levels of abstraction. Besides this we consider that generalization is different from refinement, which implies that refinement cannot be represented through a generalization relationship (e.g. the use case *Borrow Book* can be specialized into *Borrow Book to Student* and *Borrow Book to Teacher*; the use case *Borrow Book* can be refined into *Request Book Borrowing* and *Return Borrowed Book*; despite the request and the return happening in different points in time, both are needed in order to fulfill a book borrowing, which means that a book cannot be borrowed without requesting it and without returning it).

Fowler made the following advice in his book "UML Distilled" [12]: "don't try to break down use cases into sub-use cases and subsub-use cases using functional decomposition. Such decomposition is a good way to waste a lot of time". We cannot agree with Fowler's opinion at a certain extent. The pertinence of functional decomposition lies on the scale of the software system under development. The development of large software systems benefits from decomposing the functionality of those systems to a level that allows delivering less complex modeling artifacts to the teams implementing the software system. All the more large software systems are frequently built from a series of components developed by different teams. A single team is not expected to develop the whole system, therefore it shall not be delivered the modeling artifacts concerning the whole system in order to guide the conception of the component that is required to be developed by that team [13]. Fowler made another suggestion in his book: "The UML includes other relationships between use cases beyond the simple includes, such as *«extend»*. I strongly suggest that you ignore them. I've seen too many situations in which teams can get terribly hung up on when to use different use case relationships, and such energy is wasted. Instead, concentrate on the textual description of a use case; that's where the real value of the technique lies". We completely agree with Fowler when he says that the value of use case modeling lies on the textual descriptions of use cases. The approach to use case refinement we are presenting in this paper is based on those descriptions. But we cannot agree when Fowler says that the relationships besides the *«include»* relationship shall be ignored when modeling use cases. The relationship we are proposing in this paper (the *«refine»* relationship) cannot be ignored. It is needed in order to formalize at an early stage (the use case modeling) where functional decomposition shall happen in order to decrease the complexity of the modeling artifacts delivered to the different development teams.

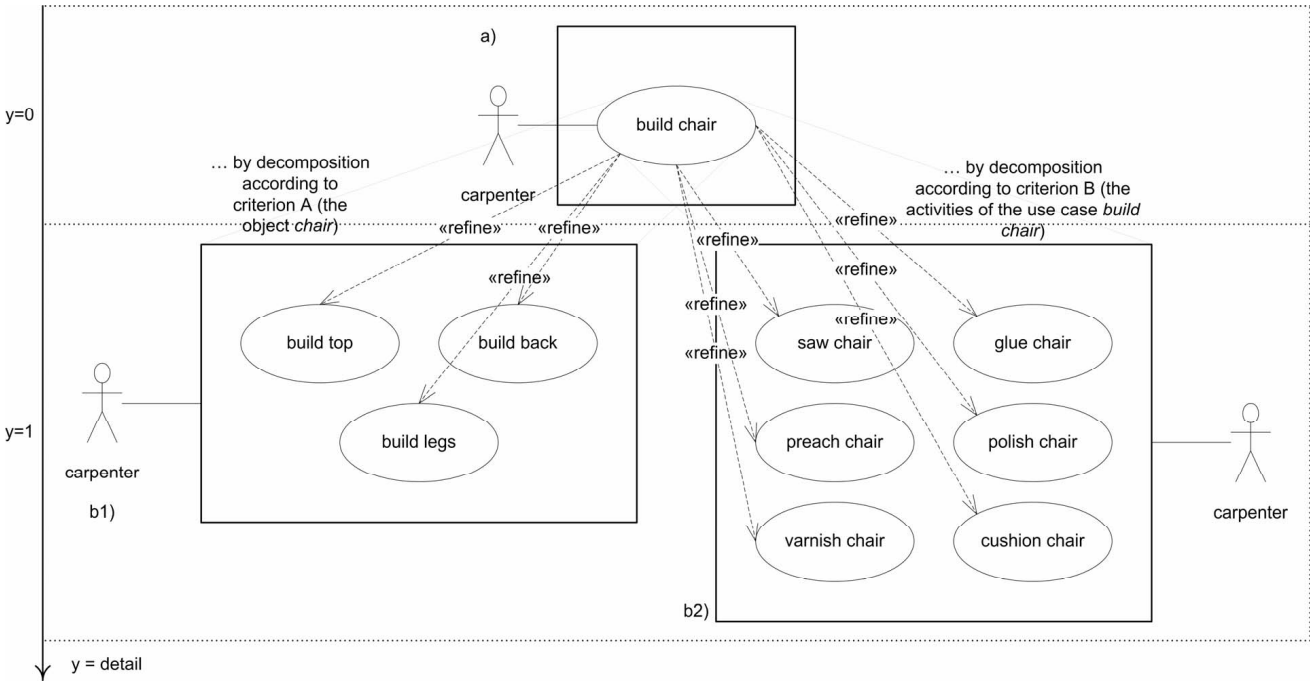

Figure 1. Refinement by decomposition according to criterion A and by decomposition according to criterion B.

III. USE CASE MODELING FROM THE PERSPECTIVE OF DETAIL

Detail in the context of this approach is intimately related to the activity of use case refinement. In this sense use cases can be more detailed if they are refined. By refining use cases the artifacts resulting from the refinement process (the refining use cases) are situated in lower abstraction levels comparatively to the refined use cases (the use cases that were submitted to the refinement process). In order to represent in the use case diagram this decrease in the abstraction level when refining use cases the *«refine»* relationship is used. The refinement process of use cases can be represented by a tree-like form that in terms of detail presents use cases hierarchically, being the more abstract ones at the top and the more concrete ones at the bottom.

Although use case diagrams are part of the UML (which follows the object-oriented paradigm) there is no restriction for the applicability of our approach to the development of software according to other software development paradigms (e.g. the functional paradigm). For instance, data-flow diagrams can also be refined [14].

IV. THE *«INCLUDE»* AND THE *«REFINE»* RELATIONSHIPS

The *«include»* relationship involves two types of use cases: the *including* use case (the use case that includes other use cases) and the *included* use case (the use case that is included by other use cases). In the context of the *«include»* relationship the UML Superstructure states that the including use case depends on the addition of the included use cases to be complete. Nevertheless in our opinion the functionality of the included use cases shall be described in the including use case. Since we rely on non-stepwise textual descriptions of use cases to determine the *«include»* relationships, the including use case has to contain the description of the included use cases so that the modeler is able to define the parts that compose the including use case (to decompose that use case). The included use case represents functionality common to various (including) use cases. But the *«include»* relationship may be used to partition the including use case into two or more use cases at the same level of abstraction instead of being used to evidence functionality common to various use cases. In that case the *«include»* relationship is used to decompose the including use case without detailing it, so the sum of the functionality represented by the non-stepwise textual descriptions of the included use cases shall be equal to the functionality represented by the non-stepwise textual description of the including use case (excluding glue logic), which implies having two or more included use cases for a single including use case.

Refinement can be defined *by decomposition according to criterion A* or *by decomposition according to criterion B*. *Refining* a use case *by decomposition according to criterion A* produces lower-abstraction-level use cases by detailing the use case and splitting it according to the parts that compose the object of that use case. In the example shown in Figure 1 the object (*chair*) is the whole and the objects *top*, *back* and *legs* are the parts of that whole, therefore refining the use case *build chair* equaled splitting it into the use cases *build top*, *build back* and *build legs*. *Refining* a use case *by decomposition according to criterion B* equals splitting the use case into activities, which also results in lower-abstraction-level use cases by detailing the use case and splitting it according to the activities that compose the use case being split. Figure 1 illustrates the refinement of the use case *build chair* by decomposition according to criterion B

by splitting it into the use cases *saw chair*, *glue chair*, *preach chair*, *polish chair*, *varnish chair* and *cushion chair* as the activity of building includes the activities of sawing, gluing, preaching, polishing, varnishing and cushioning. Although the use case under refinement is split into two or more use cases, resembling the decomposition of use cases through the *«include»* relationship (or even the (dis)aggregation/(de)composition of use cases; the generalization of use cases can also resemble refinement), the abstraction level decreases as the use cases that refine the use case under refinement are more detailed than it is. This is the distinction between the *«include»* relationship (and also the aggregation/composition association and the generalization relationship) and the refinement relationship (*«refine»*) that we will present ahead in this section of the paper. The *«refine»* relationship implies that the result of executing the more detailed use cases together shall be equal to the result of executing the less detailed use case.

In the context of classes some stereotypes (which are part of the standard UML stereotypes [2]) deal with refinement. The stereotype *«refine»* (which is applicable to the *Abstraction* dependency) represents a unidirectional or bidirectional relationship between diagram elements at different levels of abstraction (e.g. analysis and design levels). The *Abstraction* dependency represents a relationship that relates two elements representing the same concept at different levels of abstraction or from different viewpoints. It also represents a dependency in which there is a mapping between the supplier and the client. A class at the analysis level may map to more than one class at the design level, which means that we can have a set of client elements for a single supplier element. We do not recommend using the *Abstraction* dependency to represent refinement of use cases because it can be bidirectional (and refinement is unidirectional).

In the UML Superstucture [2] (in the context of use cases, particularly in the description of the semantics) the *«include»* relationship is stated to be used for the purpose of extracting the common part of the functionality of two or more use cases to a separate use case to be included (or reused) by those two or more use cases. It may be the case that we want to replace (in a lower abstraction level) a use case by two or more detailed use cases. Figure 1 depicts such situation (1a is less detailed than 1b1 and than 1b2). In this case we will end up with two use case diagrams, the later more detailed than the previous one. For this argument we consider that the use of the system represented by the use case in Figure 1a represents the uses of the system that the use cases in Figure 1b1 and that those in Figure 1b2 represent as well. The difference is that the use case in Figure 1a is less detailed than the use cases in Figure 1b1 together and the use cases in Figure 1b2 together as well. We do not recommend using the *«include»* relationship to represent the lowering of use cases' abstraction level since it is not according to its semantics in the UML metamodel.

We propose an extension to the UML metamodel to make available a UML relationship to be used in the context of use cases for representing their refinement. Figure 2 illustrates a new UML metaclass (the *Refine* metaclass) we have created to satisfy the need for extension of the UML metamodel we have identified. As far as the unidirectional association is concerned, the end named *detail* references the more detailed use case (the refining use case) and the association means that one or more *Refine* relationships refer to one (more detailed) use case. Regarding the aggregation, the end named *refine* references the *Refine* relationships owned by the use case and the end named *refinedCase* references the use case that has been detailed (the refined use case) and owns the *Refine* relationship. The metamodel tells us that two or more *Refine* relationships are owned by one (refined) use case, and one (refined) use case may be detailed and own two or more *Refine* relationships. Summarily a refined use case shall be refined by more than one refining use case and a refining use case shall refine one or more refined use cases (more than one refined use case if the refined use cases are connected through *«include»* relationships; see Figure 3 for an OCL [15] constraint on this). We wrote an OCL constraint (in Figure 4) for expressing the impossibility of having two use cases connected by both an *«include»* relationship and a *«refine»* relationship since the first does not imply increasing the detail level and the second does.

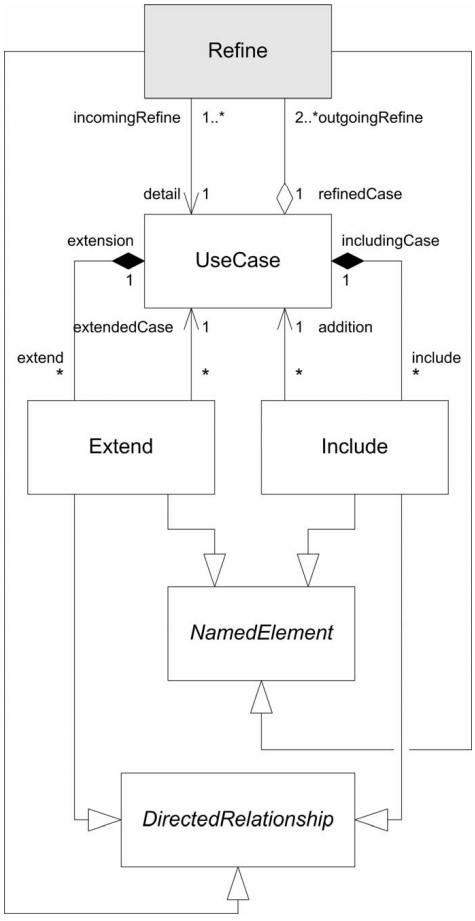

Figure 2. The proposed extension to the UML metamodel for representing the refinement of use cases.

```
context UseCase inv:
	let refines : Set(Refine) = self.incomingRefine in
	if refines->size() >= 2
	then let includes : Integer = refines->iterate(nextElement : Refine; accumulator : Integer = 0 |
accumulator->nextElement.refinedCase.include->size()) in
		refines->size() - 1 = includes
	endif
```

Figure 3. The *multiple refines* constraint.

```
context UseCase inv:
	if UseCase.include->size() >= 1
		and UseCase.refine->size() >= 1
	then UseCase.include->excludesAll(UseCase.refine)
	endif
```

Figure 4. The *coexistence* constraint.

Figure 1 exemplifies the notation of the *«refine»* relationship. It is evident by the figure that two use cases connected through a *«refine»* relationship are situated at different levels of abstraction. For instance the use cases *build top*, *build back* and *build legs* (situated at the detail level 1) are more detailed than the use case *build chair* (situated at the detail level 0). A *«refine»* relationship is represented the same way the *«include»* relationship is and from the less detailed use case to the more detailed use case in order to evidence the lowering of the abstraction level. The only difference is that the arrow is labeled with the keyword *«refine»*.

V. THE REFINEMENT PROCESS

Figure 5 illustrates how the modeler shall go from the initial use case diagram (5a) to the detailed use case diagrams (5c and 5d). It is possible to consider more than three detail levels despite we are exemplifying with three of them. The initial use case diagram (the more abstract one or less detailed one) must be analyzed independently for each of its use cases for simplicity reasons. Figure 5b shows how the partial use case diagram is elaborated from the use case 1 of the use case diagram in Figure 5a. Two *«include»* relationships have been defined for that use case, which resulted in the use cases 4 and 5. The use cases 6 and 7 are a refinement of the use case 5. That is why the use case 5 is connected to the use cases 6 and 7 through a *«refine»* relationship. The use case 4 may be refined by use cases situated at the same level of abstraction as those in the use case diagram in Figure 5c but in a distinct diagram (we have not exemplified that case due to space restrictions). The *«refine»* relationship is established between elements from two use case diagrams at different levels of detail (the partial use case diagram, the more abstract one, and the 5c use case diagram, the more detailed one). At this point it can be concluded that the *«refine»* relationship implies lowering the abstraction level (or increasing the detail level) as well as when the abstraction level decreases a new use case diagram has to be conceived. The refinement of the use case 7 (which gave origin to the use case diagram in Figure 5d) is used to show that not only included use cases or use cases that do not own any *«include»* relationship can be refined as exemplified in Figure 5. Including use cases can also be refined. We haven't exemplified that case in the figure due to space restrictions. When refining an including use case the included use cases are likely to be refined as well since their functionality is represented by the including use case as we have already explained in this paper. Figure 5 is also to depict the impossibility of having two use cases connected by both an *«include»* relationship and a *«refine»* relationship.

Figure 6 depicts two possible cases for the refinement of both an including use case and an included use case connected through an *«include»* relationship. The most adequate modeling is the one in Figure 6a where the use case 3 refines two use cases (1 and 2) and is not repeated as it is in Figure 6b. That is possible because the refined use cases are connected through an *«include»* relationship, which implies that a complete use case is repeated in two use case diagrams at the same level of abstraction (the use case diagram that refines the including use case and the use case diagram that refines the included use case).

VI. THE REFINEMENT IN THE GOPHONE CASE STUDY

The non-stepwise textual descriptions in figures 7 through 12 were elaborated based on the functional requirements from the GoPhone. As previously stated in this paper the *«include»* relationships are defined based on the non-stepwise textual descriptions of use cases. Figure 13 shows the graphical representation of the use cases textually described in figures 7 through 12. We can see that the textual descriptions of the included use cases are contained by the textual descriptions of the including use cases (e.g. the textual description of the *Compose Message* use case is contained by the textual description of the *Send Message* use case and the non-detailed textual description of the *Insert Object* use case is contained by the textual description of the *Compose Message* use case). This is an evidence of how *«include»* relationships imply decomposition but no detailing (of the including use cases' textual descriptions). The *«refine»* relationships imply that the textual descriptions of the refining use cases are more detailed than the textual descriptions of the refined use cases and also that for a single refined use case we have more than one refining use case (which means that *«refine»* relationships imply decomposition besides detailing). For instance the textual description of the *Browse Directory* use case is contained by the detailed textual description of the *Insert Object* use case

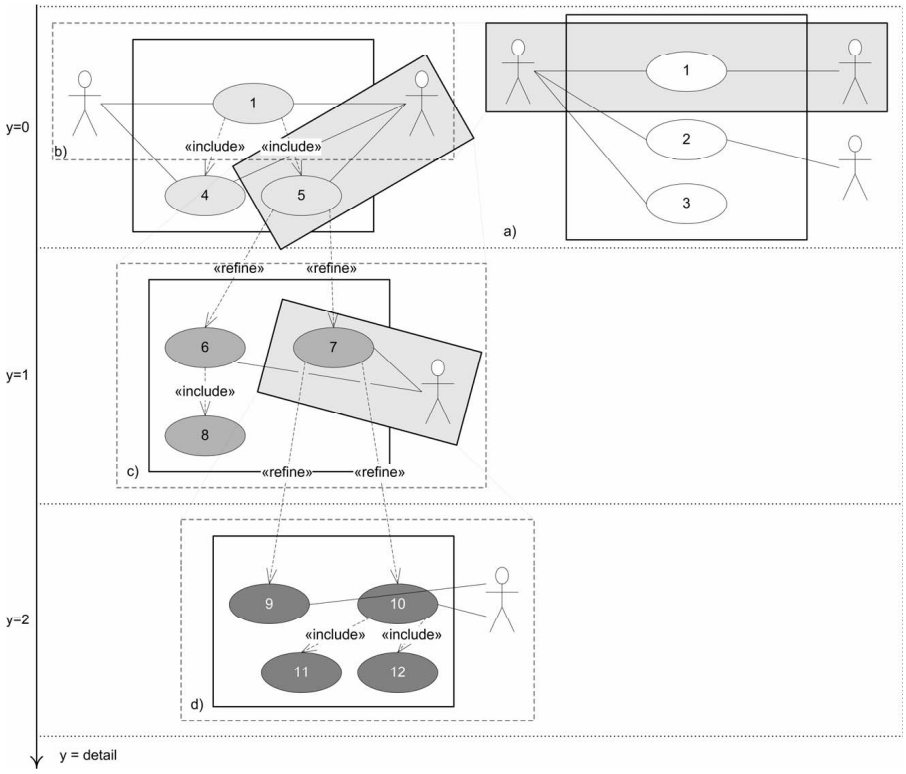

Figure 5. The refinement process.

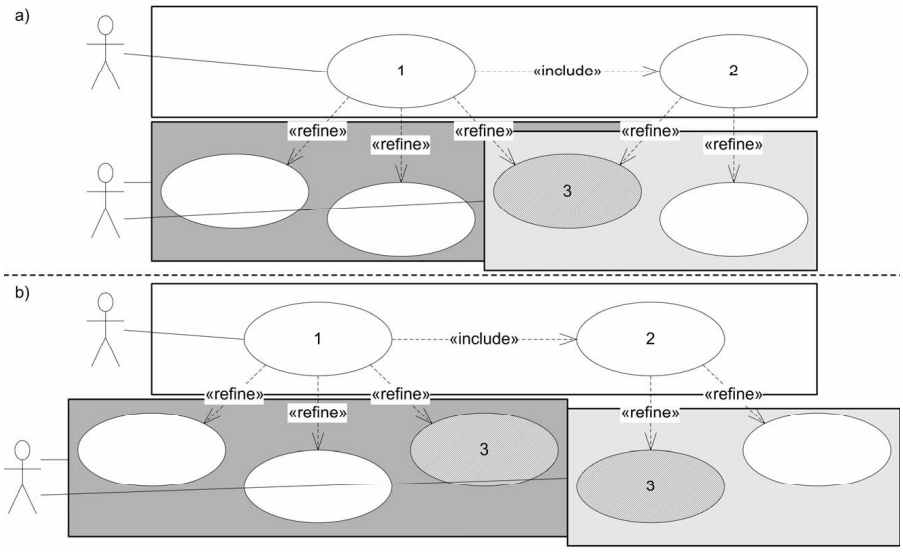

Figure 6. Possibilities for the refinement of both an including use case and an included use case.

(note that this detailed textual description is not the description corresponding to the *Insert Object* use case in the use case diagram, rather the non-detailed textual description of the *Insert Object* use case is; the more detailed textual description was only used as an intermediary/auxiliary means to get to the descriptions of the refining use cases *Browse Directory* and *Display Object in Message Area*). This is an evidence that the use cases at the detail level 1 in the figure are more detailed than the use cases at the detail level 0 in the figure.

The sum of the functionality represented by the non-stepwise textual descriptions of the included use cases shall be equal to the functionality represented by the non-stepwise textual description of the including use case. In the use case diagram at the detail level 0 in Figure 13 although the sum of the functionality represented by the non-stepwise

Send Message: The mobile user can send some different kinds of messages through the GoPhone: basic SMS, extended SMS and e-mail. Upon request from the GoPhone the mobile user chooses which kind of message he wants to send. The mobile user writes the message in a text editor. When writing the message the mobile user may activate letter combination (T9). The mobile user can insert objects into a message: pictures and/or draft texts. The mobile user can attach objects to a message: files, business cards, calendar entries and/or sounds. The mobile user chooses the recipient's contact: phone number or e-mail address depending on the kind of message (phone number in the case of a SMS, e-mail address in the case of an e-mail). The GoPhone connects to the network to send the message. In order for the GoPhone to show an acknowledgement to the mobile user (stating that the message was successfully sent) it has to receive an acknowledgement from the network. The mobile user may choose to save the message into the sent messages folder.

Figure 7. Non-stepwise textual description of the use case *Send Message*.

Compose Message: The mobile user can send some different kinds of messages through the GoPhone: basic SMS, extended SMS and e-mail. Upon request from the GoPhone the mobile user chooses which kind of message he wants to send. The mobile user writes the message in a text editor. When writing the message the mobile user may activate letter combination (T9). The mobile user can insert objects into a message: pictures and/or draft texts. The mobile user can attach objects to a message: files, business cards, calendar entries and/or sounds. The mobile user chooses the recipient's contact: phone number or e-mail address depending on the kind of message (phone number in the case of a SMS, e-mail address in the case of an e-mail).

Figure 8. Non-stepwise textual description of the use case *Compose Message*.

Insert Object: The mobile user can insert objects into a message: pictures and/or draft texts. The mobile user may receive notifications on the violation of validation rules over the objects to be inserted into the message.

Figure 9. Non-detailed non-stepwise textual description of the use case *Insert Object*.

Insert Object: The mobile user selects the objects from a repository of objects (eventually with folders), which he can browse. Upon selection of the objects from the repository they are displayed to the mobile user in the message area of the message editor. The mobile user may receive notifications on the violation of validation rules over the objects to be inserted into the message. The violation of those rules prevents the display of the invalid objects to the mobile user.

Figure 10. Detailed non-stepwise textual description of the use case *Insert Object.*

Browse Repository: The mobile user selects the objects from a repository of objects (eventually with folders), which he can browse.

Figure 11. Non-stepwise textual description of the use case *Browse Directory*.

Display Object in Message Area: Upon selection of the objects from the repository they are displayed to the mobile user in the message area of the message editor. The mobile user may receive notifications on the violation of validation rules over the objects to be inserted into the message. The violation of those rules prevents the display of the invalid objects to the mobile user.

Figure 12. Non-stepwise textual description of the use case *Display Object in Message Area.*

textual descriptions of the use cases included by the *Send Message* use case is equal to the functionality represented by the non-stepwise textual description of the *Send Message* use case, the actor *Mobile User* was not associated with the *Send Message* use case but it could have been. We did not do that because we wanted to explicitly evidence the actor of each one of the included use cases in particular since there are two actors involved in the *Send Message* use case (the *Mobile User* and the *Network*). That is not what happens with the *Compose Message* use case as there is only one actor involved in the use case.

Regarding the use case diagram at the detail level 1 in Figure 13 we can see that the refining use cases in there are associated with an actor, which means that refining use cases have to be utilizations of the system by themselves (all use cases shall have an association with the exterior of the system they belong to whether they are including, included, refined or refining use cases, otherwise we wouldn't be talking about use cases).

VII. CONCLUSIONS

In this paper we have elaborated on how the UML does not support refinement of use cases at the moment and how it can be extended in order to support that formally. As a result we have proposed to extend the UML metamodel with a new kind of relationship in the context of use cases (the *«refine»* relationship). The support of use case refinement is pertinent in large software systems development in order to deliver less complex modeling artifacts to the teams implementing those systems. Use cases shall be delivered to the different teams with responsibility for further designing and implementing the different sets of functionalities (a single team is not expected to develop the whole system). According to what was clarified in this paper the *«include»* relationship is not appropriate to model the refinement of use cases since the refinement activity implies lowering the abstraction level of use cases (particularly of their non-stepwise textual descriptions). Despite this the *«include»* relationship shall not be discarded and shall live along with the *«refine»* relationship as this paper elucidated. With this

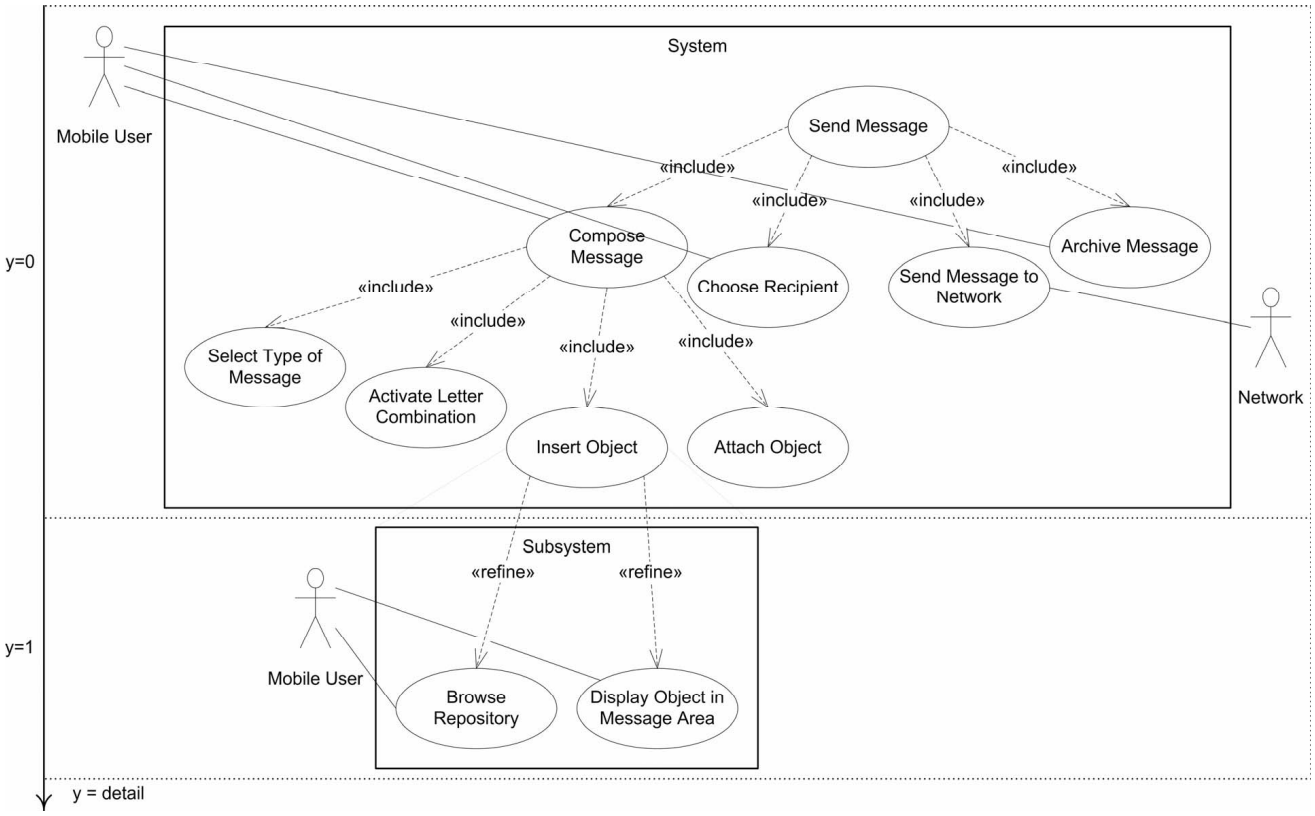

Figure 13. The use case model of the *Send Message* functionality from the GoPhone.

paper our approach to use case modeling with support for refinement began to be exposed. Future work is to introduce other perspectives into the study: functional completeness and variability. We will expand this work on use case modeling to the field of software product lines by means of exploring the *«extend»* relationship.

REFERENCES

[1] D. Muthig, I. John, M. Anastasopoulos, T. Forster, J. Dörr, and K. Schmid, "GoPhone - A Software Product Line in the Mobile Phone Domain," Fraunhofer IESE, IESE-Report No. 025.04/E March 5 2004.

[2] OMG, "Unified Modeling Language: Superstructure - version 2.2," Object Management Group, 2009, pp. 740.

[3] B. Paech and B. Rumpe, "A New Concept of Refinement used for Behaviour Modelling with Automata," in *2nd International Symposium of Formal Methods Europe (FME 1994)*. Barcelona, Spain: Springer-Verlag, 1994.

[4] D. A. C. Quartel, L. F. Pires, H. M. Franken, and C. A. Vissers, "An Engineering Approach towards Action Refinement," in *5th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS 1995)*. Chenju, Korea: IEEE Computer Society, 1995.

[5] R. Darimont and A. v. Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration," in *4th Symposium on the Foundations of Software Engineering (FSE-4)*. San Francisco, California, USA: ACM, 1996.

[6] M. Schrefl and M. Stumptner, "Behavior Consistent Refinement of Object Life Cycles," in *16th International Conference on Conceptual Modeling (ER 1997)*. Los Angeles, California, USA: Springer-Verlag, 1997.

[7] B. Mikolajczak and Z. Wang, "Conceptual Modeling of Concurrent Systems through Stepwise Abstraction and Refinement Using Petri Net Morphisms," in *22nd International Conference on Conceptual Modeling (ER 2003)*. Chicago, Illinois, USA: Springer-Verlag, 2003.

[8] D. Batory, J. N. Sarvela, and A. Rauschmayer, "Scaling Step-Wise Refinement," *IEEE Transactions on Software Engineering*, vol. 30, pp. 355-371, 2004.

[9] S. S.-s. Cherfi, J. Akoka, and I. Comyn-Wattiau, "Use Case Modeling and Refinement: A Quality-Based Approach," in *25th International Conference on Conceptual Modeling (ER 2006)*. Tucson, Arizona, USA: Springer-Verlag, 2006.

[10] A. Cockburn, *Writing Effective Use Cases*. Upper Saddle River, New Jersey: Addison-Wesley, 2000.

[11] C. Pons and R.-D. Kutsche, "Traceability Across Refinement Steps in UML Modeling," in *3rd UML Workshop in Software Model Engineering (WiSME 2004)*. Lisbon, Portugal: Springer-Verlag, 2004.

[12] M. Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Upper Saddle River, New Jersey: Addison-Wesley, 2004.

[13] R. J. Machado, J. M. Fernandes, P. Monteiro, and H. Rodrigues, "Refinement of Software Architectures by Recursive Model Transformations," in *7th International Conference on Product Focused Software Process Improvement (PROFES 2006)*. Amsterdam, The Netherlands: Springer-Verlag, 2006.

[14] J. M. Fernandes, J. Lilius, and D. Truscan, "Integration of DFDs into a UML-Based Model-Driven Engineering Approach," *Software and Systems Modeling*, vol. 5, pp. 403-428, 2006.

[15] OMG, "Object Constraint Language: Specification - version 2.2," Object Management Group, 2010, pp. 238.