Computer Science and Information Systems 13(1):199–216 DOI: 10.2298/CSIS151207065V

Aspects and Roles in Software Modeling: A Composition Based Comparison

Valentino Vranić and Milan Laslop

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 2, 84216 Bratislava, Slovakia
vranic@stuba.sk, milan33@gmail.com

Abstract. It's intriguing how the work on inherent aspect-oriented modeling almost completely ignores the similarity between aspect-oriented and role based decomposition and composition. Ever since the notion of aspect entered the software development arena, it has been compared to the notion of role. Findings range from identifying greater similarities to more cautious observations that albeit aspects and roles are similar, they appear to be more as complimentary with a significant effort needed to really bring them together in programming. Even a cursory comparison of Theme/UML, which represents a design part of Theme, probably the best known and most comprehensive approach to aspect-oriented modeling, to OOram, a prominent representative of approaches to role based modeling that influenced UML, reveals striking similarities in both decomposition and composition. Within a more comprehensive effort of finding the principles of a sustainable approach to aspect-oriented modeling, this paper pursues further this observation by establishing a partially reversible transformation of a Theme/UML model to the corresponding OOram model that proves principal analogy of themes to OOram collaboration view diagrams accompanied by the corresponding scenario view and interface view diagrams. An important implication is that aspects have their counterpart not in roles themselves, but in role collaboration. Based on these results, a possibility of using UML composite structure diagrams for aspect-oriented design is sketched out in the paper.

Keywords: aspect-oriented modeling, aspect, role, transformation, Theme/UML, OOram, UML, composite structure.

1. Introduction

Both aspect-oriented and role based approaches to software development build to a large extent upon object-orientation. While aspects are usually seen as a way to modularize so-called technical aspects like logging, monitoring, transaction management, or synchronization, with sometimes even a complete denial of the very existence of domain level (application specific) aspects [34], they are actually much broader concept than this [29]. Aspects provide a way of keeping concerns separate (e.g., to decouple programming language syntax and semantics [20] or in complex event processing [21, 22]) and can be used right up from early stages of software development, including specification [29], but also to conceptualize and modularize changes [3, 35, 23].

This is probably more remarkable in symmetric aspect-oriented approach where everything is decomposed into the units of one kind each of which covers only a specific aspect or view. These views recall *roles*. Roles extend functionality of objects enabling them to fit into different contexts, perform (play) the corresponding functionality, and take the effect into their inherent structure.

Roles are usually expected to have the capability of being added, removed, or replaced at runtime [15], and this is not intrinsic to aspects, though there are several approaches targeting dynamic aspect weaving [28]. However, from the perspective of modeling, this is not a critical difference. Moreover, the technology improvement or aspect-oriented implementations in dynamic environments are likely to change this in the future. It is interesting that even in their mainstream form aspects can be used—though statically—to enforce roles. The way to achieve this is embodied in the Director design pattern [25].

The close relationship of aspects and roles has been known for some time. There have been attempts to unify two concepts [15], to model aspects with roles [12], or to show how role systems are a special kind of aspect-oriented systems [14]. On the other hand, it's intriguing how the work on inherent aspect-oriented modeling almost completely ignores the similarity between aspect-oriented and role based decomposition and composition. While it is true that the similarity of Theme/UML, which represents a design part of Theme, probably the best known and most comprehensive approach to aspect-oriented modeling, to OOram, a prominent representative of approaches to role based modeling, has been observed in the initial work on Theme/UML, known at that time as composition patterns [8, 9], later work pays no attention to this [2, 7, 32].

Indeed, even a cursory comparison of Theme/UML with OOram reveals striking similarities in both decomposition and composition. This paper pursues further this observation by establishing a partially reversible transformation of a Theme/UML model to the corresponding OOram model within a more comprehensive effort of finding the principles of a sustainable approach to aspect-oriented modeling. Given the fact that OOram has had significant impact on the UML's part known as composite structure [26], this effort is likely to have a more immediate effect on aspect-oriented modeling in a broader sense of so-called advanced modularization. Such models could be based on pure UML employing its advanced features to achieve advanced modularization while being decoupled from implementation presumptions making them applicable in the context of object-oriented, aspect-oriented, or role based implementation.

The rest of the paper is organized as follows. To better understand the broader perspective of aspect-oriented programming as advanced modularization, Section 2 provides an insight into symmetry in aspect-oriented programming. Section 3, briefly explains Theme/UML by the means of an example. Using the same example, Section 4 explains OOram pointing out some similarities between OOram them Theme/UML. Sections 5 and 6 carry on the comparison further by establishing the transformation of Theme/UML models into OOram models and vice versa. Findings and their consequences to the concept of aspect and role are discussed in Section 7. Section 8 mentions some related work. Section 9 closes the paper.

2. Symmetry in Aspect-Oriented Programming

If an approach to software modeling is to serve as a basis for an effective implementation in a range of programming approaches, it has to take a broader perspective that embraces at least all known potential target implementation approaches. Aspect-oriented programming is usually identified with the AspectJ style of programming, known also as PARC aspect-oriented programming, where the aspects, as separate modularization units, affect the base object-oriented or other aspect code using sophisticated constructs, known as pointcuts, to address the points to be affected, known as join points, by the code provided in these aspects, known as advice. The complexity of such programming lies in effectively forming pointcuts out of the primitive ones. In AspectJ, this pointcut language comprises more than twenty primitive pointcuts each of which has also its annotation based variant.

In the AspectJ style of aspect-oriented programming, aspects are asymmetric with respect to the code they affect. One side of this asymmetry lies in aspects being a different kind of elements than classes. They may affect aspects, too, but some classes are necessary for a program to be executable. This is element asymmetry [16, 5, 4]. There is also relationship asymmetry: relationships among aspects and classes are defined by aspects exclusively in the form of pointcuts. Technically, in AspectJ, classes can contain pointcuts, but these can't be put into effect without the advice code.

The AspectJ style of aspect-oriented programming has undoubtedly had some influence in practice [19]. Although this style is asymmetric, inter-type declarations and advice can be used to emulate symmetric aspect-oriented programming [5], though this is probably rarely used in practice. While academic programming languages promoting symmetric aspect-oriented programming can't be said to have been ever used in practice, several popular programming languages like Scala, Ruby, or JavaScript make possible to program in a symmetric aspect-oriented way using the specific language mechanisms they bring: traits, open classes, and prototypes, respectively [5]. The importance of the interplay of asymmetric and symmetric aspect-oriented programming is apparent in use cases as a technique of choice for the initial modeling of interactive software systems (today popular user stories may be seen as their lightweight form [10]) with the extend relationship corresponding to asymmetric aspect-oriented programming and peer use cases corresponding to symmetric aspect-oriented programming [5, 17]. Consequently, it makes sense to explore the possibilities of providing a corresponding design level modeling approach that will serve as a basis for both asymmetric and symmetric aspect-oriented programming.

3. Theme/UML

Theme/UML is a UML based approach to aspect-oriented design. It is a part of the Theme approach [7], whose other part called Theme/Doc covers aspect-oriented analysis. A *theme*, a central notion to both parts of the approach, is actually a modeling representation of a concern. In Theme/UML, a theme is modeled as a package with the «theme» stereotype. Such a package may contain several structural and behavioral diagrams that together describe a particular view of a system. The overall system is obtained by theme composition.

Figure 1 shows a theme called *OrderProcessing* that is a part of the design of a web shop system. As is apparent from its name, this particular theme covers order processing.

Each order contains products selected from the product database.[1] Other concerns, such as customer sign up, traffic statistics, or even product database content management, represent separate concerns not included in this theme. Thereafter, the definitions of classes contained in the theme are partial. For example, the *Product* class contains no operations related to manipulating basic product information.

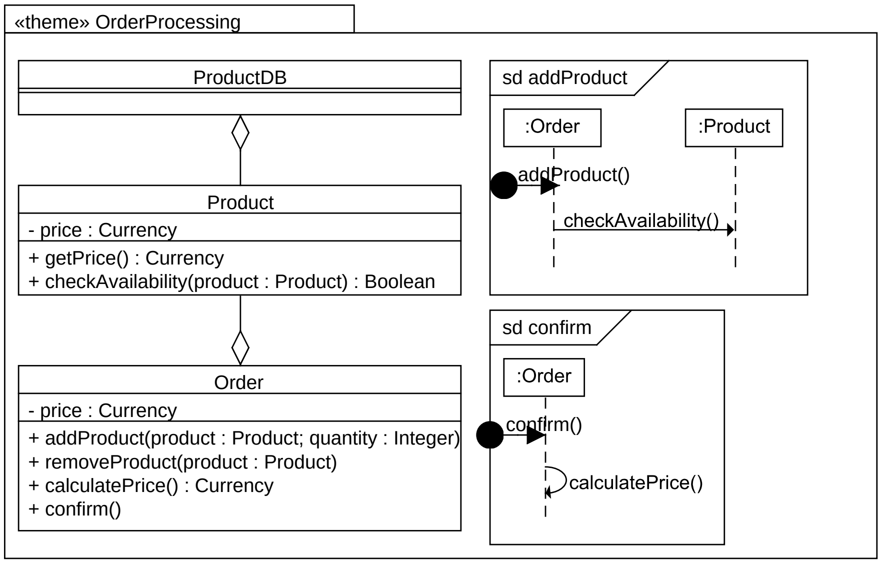

Fig. 1. A base theme.

Accordingly, behavior in themes is limited to a particular concern. Operations related to order *OrderProcessing* embrace *addProduct()* and *confirm()* that enable a product to be added to an order and to confirm that order, respectively, are presented using sequence diagrams. Themes allow for other behavioral diagrams, too, which is out of the scope of this paper.

The basic setting of order processing abstracts from the information on possible bonuses related both to adding a product and to order confirmation: a special free product is added whenever a customer adds a particular product or when the total price of the order is higher than a certain amount. Bonuses actually may be applicable in other cases, too. A general case is captured as the *BonusProviding* theme in Fig. 2. In the Theme vocabulary, this is an aspect theme, while the *OrderProcessing* theme from Fig. 1 is a base theme.

Aspect themes are parameterized and their behavior is defined with respect to formal parameters. In *OrderProcessing* there are two parameters—two operations: *BonusRequester.op()* and *BonusRequester.provideBonus()*. The first operation in the parameter list is always the trigger: an operation to be "caught" by the theme. Additional behavior is specified in a sequence diagram that redefines this operation (operation parameter lists

[1] The example is simplified and not intended to represent a comprehensive design of a web shop. As a consequence, some relevant attributes and operations are left out.

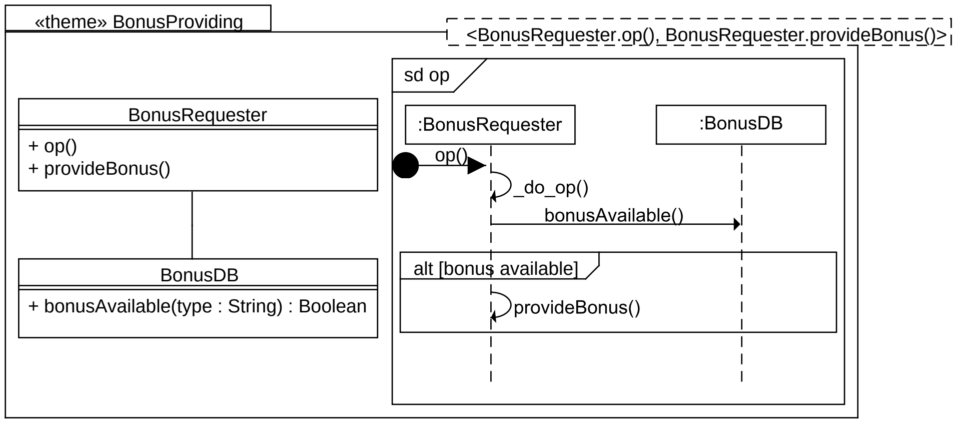

Fig. 2. An aspect theme.

are omitted for brevity). However, the original operation is preserved and available to be executed under a different name that consists of the *_do_* prefix attached to the original operation name.

The actual operations are supplied during theme composition via the *bind* clause.[2] This asymmetric aspect-oriented composition is depicted in Fig. 3.

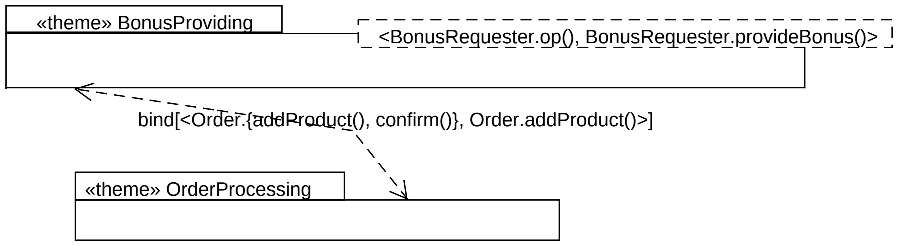
«theme» BonusProviding
<BonusRequester.op(), BonusRequester.provideBonus()>
bind[<Order.{addProduct(), confirm()}, Order.addProduct()>]
«theme» OrderProcessing

Fig. 3. A composition of an aspect theme with a base theme.

Note that any class referred to by the theme parameter list is automatically a parameter class. In Theme/UML terminology, such class is a placeholder: it just stands for a real class to be supplied during parameter binding.

The behavior prescribed by *BonusProviding* is performed both in the context of the *addProduct()* and *confirm()* operations of the *Order* class. In both cases, the *addProduct()* operation is supplied also as the operation by which the bonus is provided. Figure 4 shows how the *addProduct()* operation would look like after being affected by the *BonusProviding* theme.

[2] Parameterized types and parameter binding are actually a part of UML 2 itself, though in Theme/UML this concept is slightly adapted.

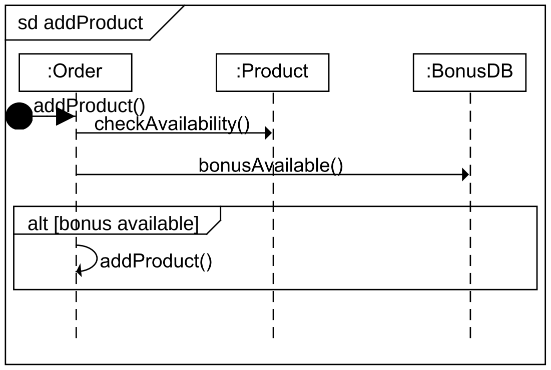

Fig. 4. The *addProduct()* operation as affected by *BonusProviding*.

Theme/UML supports symmetric aspect-oriented composition, too. Figure 5 shows this case. The *ProductDB* theme covers the fundamentals of product storing, while the details of a product are covered by the *Product* theme. Both themes are base themes, yet they are complementary and as such can be composed. In composition, themes can be merged or one theme may override another, which is indicated by the composition arrow (bidirectional or unidirectional, respectively). In this case, themes are merged, where default is to match elements by name. The resulting theme is shown in Fig. 6. Theme/UML enables full control of the composition at the level of individual class elements.

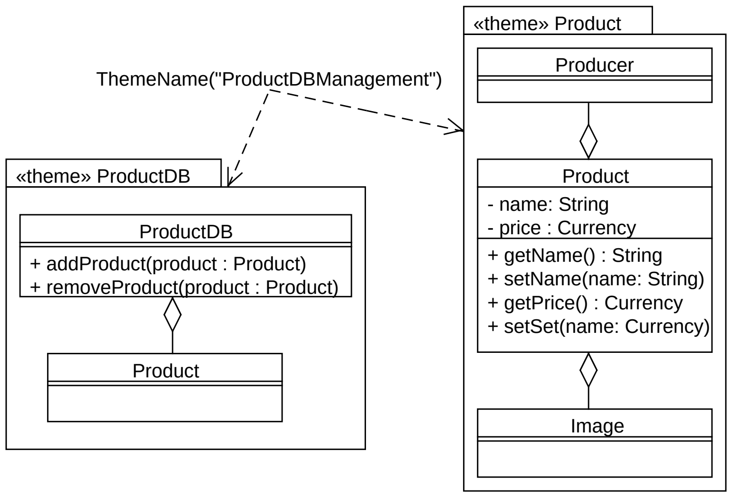

Fig. 5. A composition of two base themes.

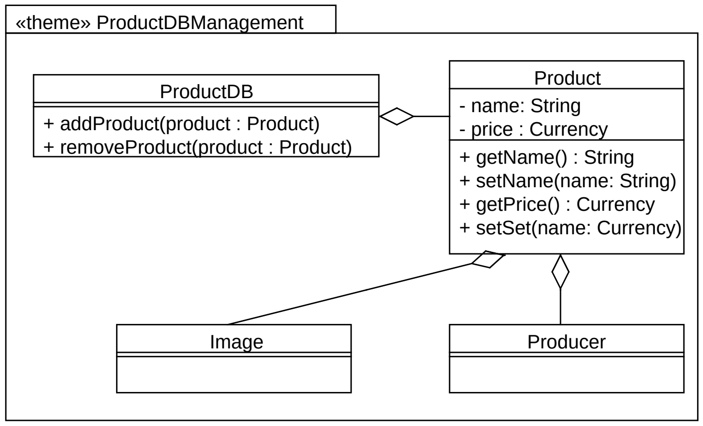

Fig. 6. A composed theme.

4. OOram

OOram—Object-Oriented Role Analysis and Modeling—is a role-based method of object-oriented software development proposed by Trygve Reenskaug, who phrased the famous Model-View-Controller pattern. OOram influenced UML to incorporate the concept of role in composite structure and the notion of collaborations [26]. Ten years prior to Theme, OOram included ideas of separating and modularizing concerns and building more complex ones via composition [31, 7].

It is important to note that OOram is not a mere modeling exercise. In fact, the actual program realization has been at its heart from the beginning [31] and the foundations laid in OOram appear as very relevant for achieving a necessary flexibility in code as demonstrated by the DCI approach (Data, Context and Interaction), in which roles and role binding are used to depart use cases from more stable parts of software systems [10, 30, 11]. This is of the utmost interest in the broader area of preserving use cases in code [6, 18, 24].

OOram employs several views of a model. One of them is the *interface view* that shows roles in collaboration with interfaces through which they collaborate. The structural part of a theme fits quite well into this representation. The classes in themes are not full-fledged classes, but partial ones containing only elements necessary with respect to the concern modularized by the theme they belong to. Figure 7 presents the structural part of the *OrderProcessing* theme from Fig. 1 rewritten as an OOram interface view. The *OrderClient* role represents a so-called environmental role, depicted as a dashed oval, which is used to express an expected client role to be defined by some further role composition. The client role is required by the OOram notation so *Order* can provide the corresponding interface to it (*Order*<*OrderClient* in this case).

Collaboration of roles is indicated by edges between them. These edges are connected to role ports, which are depicted as circles. By ports, a role expresses its ability to accept messages from another role. A simple circle indicates that the role can send messages to only one instance based on the role connected to it, while a double circle indicates that the

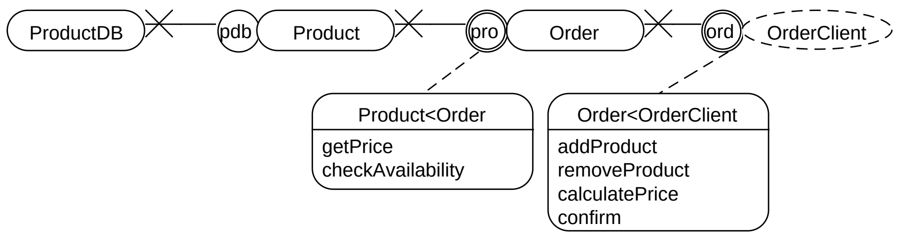

Fig. 7. An OOram interface view diagram.

role can send messages to multiple instances. Where there is no communication possible, a cross is displayed instead of a circle.

Interfaces are attached to ports. They contain operations the role expects to be provided by the role at the opposite side of the edge. For example, an *Order* expects a *Product* to provide an interface denoted as *Product<Owner* by which it would be able to perform operations *getPrice* and *checkAvailability*.

The *scenario view* shows a sequence of interactions among roles. It is quite similar to rudimentary UML sequence diagrams accompanied by chunks of text specification used to explain what is in sequence diagrams usually expressed by combined fragments, so no example is given here.

The *collaboration view* is virtually the same as the interface view without showing the actual interfaces. The composition of role models is usually expressed using the collaboration view. This composition is similar to theme composition. Figure 8 mimics the structural part of the theme composition defined in Fig. 3. Each theme corresponds to a separate model. The resulting model consists of the roles *derived* from the roles in initial models. Deriving is expressed by wide arrows on a per role basis. In general, roles can be derived from multiple initial roles. An example of this is the *BonusRequester* role derived from both *Order* and *BonusRequester*.

Of course, the structural part isn't sufficient to express the composition. The behavioral side is captured by composing scenario view diagrams. In the composition under consideration one of such composed scenario view diagrams would principally depict the same as the sequence diagram in Fig. 4. Compared to Theme/UML, in which compositions are defined via the *bind* clause, in OOram they have to be expressed explicitly.

5. Transformation from Theme/UML to OOram

Even a cursory comparison of the examples given in the previous two sections reveals that Theme/UML and OOram are similar. As we saw, both Theme/UML and OOram involve a similar process of modeling aiming at the separation of concerns. Concerns are modularized as themes in Theme/UML with each one presented (literally) as a package of the corresponding structural and behavioral models. In this, OOram relies on the notion of role model comprising of several views not necessarily displayed at once, but intended to be perceived as a whole.

Both approaches employ a similar notation. The OOram scenario view is similar to UML sequence diagrams used in Theme/UML. The OOram collaboration and interface

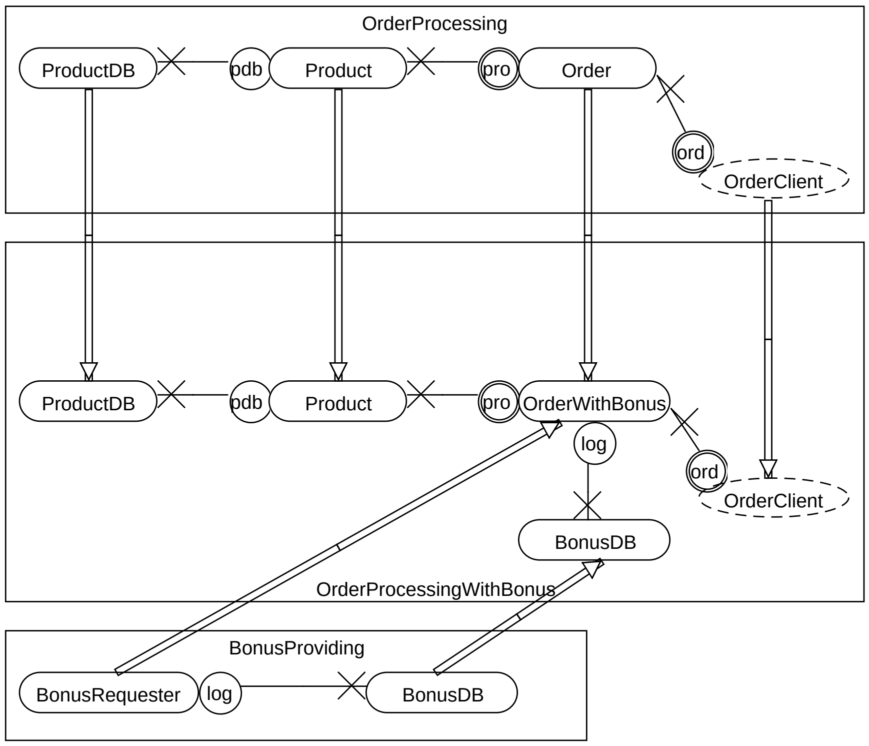

Fig. 8. Synthesis of role models.

views express the system structure and interfaces, which is similar to UML class diagrams used in Theme/UML.

In this and the following section, a more thorough comparison of Theme/UML and OOram is going to be provided by defining a transformation process of a model in Theme/UML model into an equivalent one in OOram and back. Effectively, this is going to give more practical insight on concepts of aspect and role and how they are related to each other.

The key to a correct transformation is to take elements in relationships. Let's first take a look at how could a base and aspect theme composition be transformed into OOram. Recall our web application example from Section 3. The structural part (class diagrams) of themes can be mapped to the OOram interface view as exemplified by the OOram diagram in Fig. 7 as a transformation of the *OrderProcessing* base theme from Fig. 1 and the diagram in Fig. 2 as a transformation of the *BonusProviding* aspect theme contained in Fig. 8 with suppressed interfaces. The behavioral part expressed by sequence diagrams remains practically the same (omitted here).

Next, a composition of the OOram diagrams matching the original Theme/UML composition has to be created. In creating a merged role diagram displayed in Fig. 8, the *Order* and *BonusRequester* roles have been merged into a more specific role denoted as *Order-*

WithBonus. *OrderWithBonus* is derived from both *Order* and *BonusRequester* and as such occurs in the relationships each of these roles has to other roles.

The base theme composition is transformed into OOram in a similar manner as a base and aspect theme composition. The themes are mapped to OOram diagrams the same way that has been demonstrated and the composition of two collaboration view diagrams corresponding to base themes is similar to the composition of the original base themes: the common roles are identified and the diagrams are merged, showing the common role only once.

In general, the transformation of a Theme/UML model into the corresponding OOram model involves these steps:

1. Create an interface view diagram for the structural part (class diagram) of each theme (as an example consider the OOram diagram in Fig. 7 as a transformation of the *OrderProcessing* base theme from Fig. 1 and the *BonusProviding* role collaboration in the diagram in Fig. 8, with suppressed interfaces, as a transformation of the aspect theme with the same name depicted in Fig. 2).
   (a) Create a role for each class.
   (b) Since classes tend to be wider in scope than roles, one class can be divided into several fragments according to its relationships (association) to other classes. Create a role for each such fragment.
   (c) Name each role. In the roles that correspond to whole classes, the name of the role is simply the name of the class. The names of the roles created from class fragments should reflect the relationship semantics according to which they have been created.
   (d) By this, the collaboration diagrams have been obtained. Determine the interfaces between the roles according to the methods provided by the corresponding classes. Include any environmental role as required by OOram.
2. Create a scenario view diagram for the behavioral part (sequence diagram) of each theme that describes the interaction among the roles identified in step 1
   (a) The original interactions from the sequence diagram are preserved and attached to the roles corresponding to the classes from the sequence diagram.
   (b) In the roles created by class decomposition, the proper role to appear in the scenario view diagram has to be determined according to the operations being employed.
3. For each base theme composition create the composition of the corresponding collaboration view diagrams the result of which is a new collaboration view diagram.
   (a) The correspondence of roles is determined by the correspondence of classes defined by the *merge* or *override* clause.
   (b) Consequently, the roles corresponding to the classes that correspond to each other (in the default composition this is by name, but in general, defined by the *merge* or *override* clause) will be derived from these corresponding roles. Determine their names so to reflect their combined semantics.
   (c) The roles corresponding to the non-composed classes are to be simply transferred into the resulting collaboration view diagram.
4. Create the appropriate scenario view diagrams for each new collaboration view diagram that corresponds to a base theme composition.
   (a) The unrelated operations are simply transferred as such.

   (b) For related operations, the composed scenario view diagrams have to be created according to the corresponding *merge* or *override* clause.
5. For each aspect and base theme composition create the composition of the corresponding collaboration view diagrams the result of which is a new collaboration view diagram (an example is provided in Fig. 8, which represents a transformation of the theme composition presented in Fig. 3).
   (a) The correspondence of roles is determined by the correspondence of classes defined by the *bind* clause.
   (b) Consequently, the roles corresponding to the base theme classes affected by the aspect theme will be derived from several corresponding roles. Determine their names so to reflect their combined semantics.
   (c) The unaffected roles corresponding to the base theme classes and the roles corresponding to the classes added by the aspect theme will be derived directly just from these roles preserving their names or making them specific to the new context of the collaboration view diagram being created.
6. Create the appropriate scenario view diagrams for each new collaboration view diagram that corresponds to a composition of an aspect and base theme
   (a) The unrelated operations are simply transferred as such.
   (b) For related operations, the composed scenario view diagrams have to be created according to the corresponding *bind* clause.

The transformation confirms what we have already observed from examples. If we understand a theme as an aspect in the broader sense of this notion covering both symmetric and asymmetric perspective, an aspect corresponds not to one role as could have been expected, but to a collaboration of roles.

6. Reverse Transformation

Unlike Theme/UML with its packages, OOram provides no explicit construct that will serve as a container to all the artifacts relevant to a particular concern. However, collaboration view diagrams are effectively the main entry point into the role model around which all other diagrams are based, so in our effort to define the transformation of an OOram model into the corresponding Theme/UML model, we are going to rely on this observation.

While Theme/UML records the composition in a declarative form, OOram records it in a constructed form, i.e., as another collaboration view diagram accompanied by the corresponding interface view diagram and scenario view diagrams. Thus, in a transformation from OOram to Theme/UML only initial, non-composed collaboration view diagrams should be regarded as themes. All other, composed collaboration view diagrams are actually specifications of compositions and as such have to be transformed into theme compositions.

As is probably obvious from examples, from the structural viewpoint, the OOram model obtained by the transformation of a Theme/UML model is agnostic to the asymmetric character of aspect themes. In other words, if we are to transform an OOram model into the corresponding Theme/UML model, we can't unambiguously decide whether a particular collaboration or interface view diagram should be interpreted as an aspect or base theme. However, to some extent, this can be estimated from scenario view diagrams based on how are aspect and base theme composition usually applied. Aspect themes are applied in more complicated cases heavily affecting behavioral side, while base theme composition is used mainly to interleave partial class definitions without actually composing operations with each other. Consequently, aspect theme composition tends to produce combined scenario view diagrams.

Roughly, the transformation of an OOram model into the corresponding Theme/UML model can be achieved as follows:

1. Create a theme out of each non-composed collaboration view diagram and accompanying interface view and scenario view diagrams (an example of this are the *OrderProcessing* base theme from Fig. 1, viewed as a transformation of the OOram diagram in Fig. 7, and the *BonusProviding* theme in Fig. 2, viewed as a transformation of the role collaboration with the same name depicted in Fig. 2). For each theme represented as a (non-parameterized) package, transform the collaboration view diagram and respective interface view diagrams into a class diagram (some roles may be merged into a single class) and transform each scenario view diagram into a sequence diagram.
2. Create a specification of theme composition out of each composed collaboration view diagram and accompanying interface view and composed scenario view diagrams. This will yield the corresponding merge or bind clause (an example of this can be seen in the theme composition presented in Fig. 3, viewed as a transformation of the composed collaboration view diagram depicted in Fig. 8).
3. Along with step 2, adjust the packages of the themes estimated as aspect themes to be parameterized. For each such theme, determine the parameter list based on how are the corresponding roles involved in compositions.

7. Discussion and Outlook

Table 1 summarizes the corresponding notions in Theme/UML and OOram.

Table 1. The corresponding notions in Theme/UML and OOram.

Theme/UML	OOram
theme	collaboration of roles
parameter class in an aspect theme	role
non-parameter class in an aspect theme	role
class	role or collaboration of roles
class fragment	role
operation	interface method
bind	two roles relationship
base theme	collaboration view diagram
aspect theme	collaboration view diagram
concept sharing	role sharing in the collaboration diagram
crosscutting	relationship between two roles
decomposition: theme creation	decomposition: role model creation
composition: composing themes	synthesis: composing role diagrams
structural diagram (class diagram)	collaboration/interface view diagram
behavioral diagram (sequence diagram)	scenario view diagram

When comparing aspects and roles in isolation according to their main properties, an aspect might seem to correspond to a role. However, the comparison based on the transformation of models clearly showed that if we understand a theme as an aspect in the broader sense of this notion covering both symmetric and asymmetric perspective, an aspect corresponds to a collaboration of roles, which is represented by a collaboration view diagram in Theme/UML. Aspect composition then corresponds to the composition of role collaborations.

The full interchangeability of OOram and Theme/UML would have been proven if a reversible transformation could have been established. A reversible transformation (the original model can be reconstructed from the one it has been transformed into) would guarantee that no information is lost during the transformation confirming that both approaches have an equal power of expression. However, the transformation shown in this paper can't be considered fully reversible, because the information whether a composition is symmetric or asymmetric is lost in OOram and when performing the reverse transformation (from OOram back to Theme/UML) this information can be reconstructed only with some probability.

To some extent, the tendency of asymmetric aspect-oriented composition to affect many places at once, which is known as quantification, can be of help in deciding the symmetry of role composition. This can be observed even at the structural level: if the matching role in the composition at some level of specialization has a relationship to several different roles, then probably an asymmetric composition is in case. However, even if this doesn't hold, the asymmetric composition can still be used, e.g., to prepare a crosscutting concern involved only in one place to be applicable to other places, too.

With respect to quantification, it is worthwhile noting that OOram lacks any kind of quantification mechanism, while Theme/UML supports these by enabling multiple element specification [7], which is based on using regular expressions in name signatures in a similar way as they are is used in AspectJ.

It is worthwhile observing that the composite structure diagrams in UML strongly recall collaboration view and interface view diagrams of OOram. This can be a basis for a common approach to expressing role and aspect-oriented models in a standard UML. Figure 9 depicts the composition of role models captured as collaborations. In UML, a *collaboration*, depicted as a dashed ellipse, describes some aspect of the cooperation of certain instances, depicted as rectangles [27]. The instances in this case represent *roles*. A role can be derived from another role. A role derived from more than one role consequently represents their composition.

Of course, the actual role collaboration has to be defined by interaction diagrams (such as sequence diagrams). A role collaboration defined this way can be transferred to specific objects by role binding. In case of simultaneous binding of an object to several roles, role binding actually represents another possibility of defining role composition. This is still a graphical composition that may be not as flexible as the textual one. In this sense, the possibilities of using OCL should be explored. This way of modeling could be applicable also in the DCI approach (mentioned in Section 4) or in the envisioned 3D rendering of UML [13].

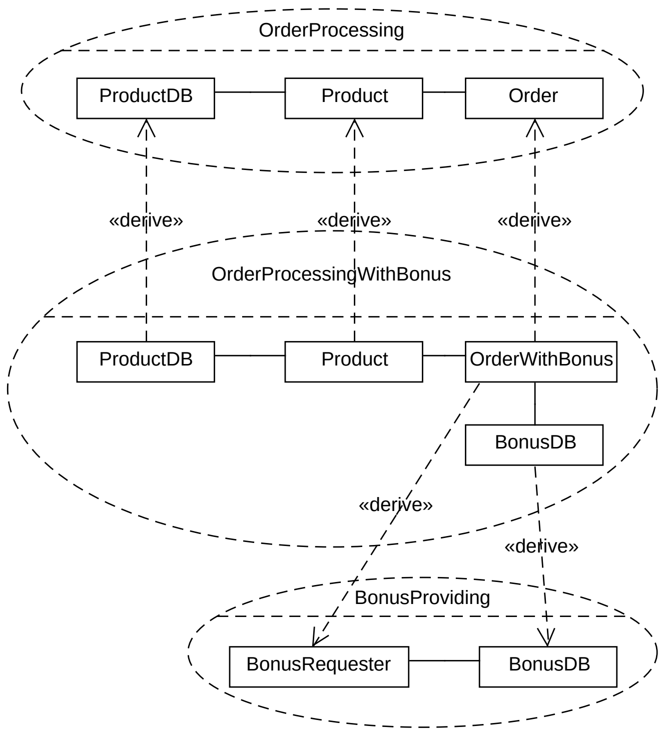

Fig. 9. Composition of role models in UML.

8. Related Work

As has been mentioned in the introduction, the similarity of Theme/UML to OOram has been observed in the initial work on Theme/UML [8, 9]. Theme merging in Theme/UML, i.e., symmetric aspect-oriented composition, has been related to role composition in OOram, but no further elaboration of this has been provided, neither the themes have been explicitly related to roles.

Hanenberg et al. base their assessment of roles from the aspect-oriented perspective in how both aspect and roles may enhance a given set of objects with additional features and may influence their dynamic behavior [14], which corresponds to the asymmetric understanding of aspects. This paper takes a broader perspective that incorporates the symmetric aspect-oriented approach.

Hanenberg and Unland—again in the asymmetric manner—compare characteristics of aspects and roles concluding they are not the same [15]. They also explored how aspects and roles can be used together in AspectJ finding this to be of a questionable usability.

Steimann also relates aspects to individual roles [33, 34], and not to collaborations of roles. He finds the two notions to be different since aspects are applied individually and obliviously, while roles are applied as collaborations. While aspects indeed are sometiems applied in an opportunistic fashion and by a programmer oblivious of the rest of the application, as Rashid and Moreira point out, obliviousness is not an essential property of aspects [29].

Graversen and Østerbye argue that a role can't define an aspect since it is a crosscutting concern and as such affects multiple entities [12]. This is known as quantification and has also been disputed as an essential property of aspects [29]. Nevertheless, Graversen and Østerbye claim that because of inability of roles to affect multiple entities, a set of roles is necessary to define an aspect. This is exactly opposite to the findings of this paper where an aspect is seen as a collaboration of roles.

The comparison of Theme/Doc, the analytical part of the Theme approach, with use cases [36] has revealed that analytical themes correspond to use cases. This is in accordance with Jacobson and Ng's approach of aspect-oriented software development based on use cases [17]. While Jacobson and Ng do carry use cases a step further in modeling and capture their realization as collaborations, which represent a concept that belongs to composite structure mentioned in the previous section, they don't discuss the possibility of the composition of these collaborations.

Apel et al. [1] compare the possibilities of implementing aspects as roles with respect to the asymmetric style of aspect-oriented implementation. They conclude that aspects in asymmetric aspect-oriented programming can serve as role implementations both on individual basis, i.e., one aspect per role, or having each collaboration of roles captured by an aspect, finding the latter to be very similar to their so-called feature oriented programming language Jak, which embraces elements of role based programming. Jak is actually an attempt at providing symmetric aspect-oriented programming, as the authors themselves admit:

> Note that we do not consider symmetric AOP approaches [57] (for example, subject-oriented programming [56], [92] or aspectual components [68]) since they are much closer (if not similar) to our notion of FOP than to our notion of AOP.

This is consistent with our findings regarding the relationship between aspect and role in modeling as a higher level of abstraction, which—referring back to programming—can be stated as follows: aspects in both asymmetric and symmetric aspect-oriented programming can be seen as role collaborations.

In our discussion of symmetry in aspect-oriented programming, we haven't taken into account join point symmetry. As conceived by Harrison et al. [16], join point symmetry is defined only in the sense of static aspect-oriented composition (that can be performed on lexical basis), so it is of a limited applicability to contemporary aspect-oriented approaches [5].

9. Conclusion and Further Work

This study provided a new insight into the relationship between aspects and roles from the perspective of modeling. Two relevant approaches to aspect-oriented modeling and role based modeling, Theme/UML and OOram (Object-Oriented Role Analysis and Modeling), have been compared in an attempt to establish a reversible transformation. Albeit this has revealed to be not fully possible, substantial similarities between the two approaches have been confirmed with respect to decomposition and composition.

The main difference between Theme/UML and OOram is that symmetry or asymmetry of composition, which is supported explicitly in Theme/UML, disappears in OOram. Another important difference is that Theme/UML is more declarative in representing composition making it easier to maintain it, while OOram is very explicit in its graphical expression of composition based on resulting (composed) diagrams. Despite these differences, the study presented in this paper brings us to the conclusion that aspects both in their symmetric and asymmetric understanding have their counterpart not in the roles themselves, but in role collaboration.

Yet another important finding is that composite structure diagrams in UML strongly recall collaboration view and interface view diagrams of OOram. This a logical direction for further work: reestablishing aspect-oriented modeling at the design stage on the basis of composite structure diagrams.

Acknowledgments. The work reported here was supported by the Scientific Grant Agency of Slovak Republic (VEGA) under the grant No. VG 1/1221/12. This contribution/publication is also a partial result of the Research & Development Operational Programme for the project Research of Methods for Acquisition, Analysis and Personalized Conveying of Information and Knowledge, ITMS 26240220039, co-funded by the ERDF.

References

1. Apel, S., Leich, T., Saake, G.: Aspectual feature modules. IEEE Transactions on Software Engineering 34(2), 162–180 (2008)
2. Baniassad, E.L.A., Clarke, S.: Theme: An approach for aspect-oriented analysis and design. In: Proceedings of the 26th International Conference on Software Engineering, ICSE 2004. Edinburgh, UK (2004)
3. Bebjak, M., Vranić, V., Dolog, P.: Evolution of web applications with aspect-oriented design patterns. In: Brambilla, M., Mendes, E. (eds.) Proceedings of ICWE 2007 Workshops, 2nd International Workshop on Adaptation and Evolution in Web Systems Engineering, AEWSE 2007, in conjunction with 7th International Conference on Web Engineering, ICWE 2007. pp. 80–86. Como, Italy (Jul 2007)
4. Bálik, J., Vranić, V.: Sustaining composability of aspect-oriented design patterns in their symmetric implementation. In: 2nd International Workshop on Empirical Evaluation of Software Composition Techniques, ESCOT 2011, at ECOOP 2011. Lancaster, UK (2011)
5. Bálik, J., Vranić, V.: Symmetric aspect-orientation: Some practical consequences. In: Proceedings of NEMARA 2012: International Workshop on Next Generation Modularity Approaches for Requirements and Architecture, at AOSD 2012. ACM, Potsdam, Germany (2012)
6. Bystrický, M., Vranić, V.: Preserving use case flows in source code. In: Proceedings of 4th Eastern European Regional Conference on the Engineering of Computer Based Systems, ECBS-EERC 2015. IEEE, Brno, Czech Republic (2015)
7. Clarke, S., Baniassad, E.: Aspect-Oriented Analysis and Design: The Theme Approach. Addison-Wesley (2005)
8. Clarke, S., Harrison, W., Ossher, H., Tarr, P.: Subject-oriented design: Towards improved alignment of requirements, design and code. In: Proceedings of 14th ACM SIGPLAN Conference Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA'99. Denver, Colorado, USA (1999)
9. Clarke, S., Walker, R.J.: Composition patterns: An approach to designing reusable aspects. In: Proceedings of 23rd International Conference on Software Engineering, ICSE-23. pp. 5–14. Toronto, Canada (2001)

10. Coplien, J., Bjørnvig, G.: Lean Architecture: for Agile Software Development. Wiley (2010)
11. Coplien, J.O., Reenskaug, T.: The data, context and interaction paradigm. In: Proceedings of 3rd Annual Conference on Systems, Programming, and Applications: Software for Humanity, SPLASH '12. pp. 227–228 (2012)
12. Graversen, K.B., Østerbye, K.: Aspect modelling as role modelling. In: Proceedings of OOPSLA '02 Workshop on Tool Support for Aspect Oriented Software Development (2002)
13. Gregorovič, L., Polášek, I., Sobota, B.: Software model creation with multidimensional UML. In: Proceedings of of International Conference on Research and Practical Issues of Enterprise Information Systems, CONFENIS 2015, 23rd IFIP World Computer Congress International Conference on Research and Practical Issues of Enterprise Information Systems. LNCS 9357, Springer, Daejeon, Korea (2015)
14. Hanenberg, S., Stein, D., Unland, R.: Roles from an aspect-oriented perspective. In: Proceedings of VAR'05: Views, Aspects and Roles Workshop, ECOOP 2005. Glasgow, UK (2005)
15. Hanenberg, S., Unland, R.: Roles and aspects: Similarities, differencies, and synergetic potential. In: Proceedings of 8th International Conference on Object-Oriented Information Systems, OOIS 2002. Montpellier, France (Sep 2002)
16. Harrison, W.H., Ossher, H.L., Tarr, P.L.: Asymmetrically vs. symmetrically organized paradigms for software composition. Tech. Rep. RC22685, IBM Research (Dec 2002)
17. Jacobson, I., Ng, P.W.: Aspect-Oriented Software Development with Use Cases. Addison-Wesley (2004)
18. Khelifi, N.Y., Śmiałek, M., Mekki, R.: Generating database access code from domain models. In: Proceedings of 2015 Federated Conference on Computer Science and Information Systems, FedCSIS 2015, 5th Workshop on Advances in Programming Languages, WAPL 2015. IEEE, Łódź, Poland (2015)
19. Laddad, R.: A real-world perspective of aop. Transactions on Aspect-Oriented Software Development VIII (2011)
20. Laddad, R.: Abstract syntax driven approach for language composition. Central European Journal of Computer Science 4(3), 107–117 (2014)
21. Lang, J., Jantošovič, M., Polášek, I.: Re-usability in complex event pattern monitoring. In: Proceedings of IEEE 10th Jubilee International Symposium on Aplied Machine Intelligence and Informatics, SAMI 2012. IEEE, Herľany, Slovakia (2012)
22. Lang, J., Jánik, J.: Reactive distributed system modeling supported by complex event processing. In: Proceedings of 3rd Eastern European Regional Conference on the Engineering of Computer Based Systems, ECBS-EERC 2013. IEEE Computer Society, Budapest, Hungary (2013)
23. Menkyna, R., Vranić, V.: Aspect-oriented change realization based on multi-paradigm design with feature modeling. In: Proceedings of 4th IFIP TC2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2009, Revised Selected Papers. LNCS 7054, Springer, Krakow, Poland (2012)
24. Śmiałek, M., Nowakowski, W., Jarzębowski, N., Ambroziewicz, A.: From use cases and their relationships to code. In: Proceedings of 2nd IEEE International Workshop on Model-Driven Requirements Engineering, MoDRE 2012. IEEE, Chicago, IL, USA (2012)
25. Miles, R.: AspectJ Cookbook. O'Reilly (2004)
26. Møller-Pedersen, B.: Scandinavian contributions to object-oriented modeling languages. In: History of Nordic Computing 3 – 3rd IFIP WG 9.7 Conference, HiNC 3, Revised Selected Papers. IFIP, Stockholm, Sweden (2010)
27. Object Management Group: OMG Unified Modeling Language (OMG UML) Superstructure, Version 2.4.1. OMG (Aug 2011), formal/2011-08-06
28. Okanović, D., Vidaković, M.: Evaluation of alternative instrumentation frameworks. In: Proceedings of the Symposium on Software Performance: Joint Descartes/Kieker/Palladio Days, SOSP 2014. Stuttgart, Germany (Nov 2014)

29. Rashid, A., Moreira, A.: Domain models are NOT aspect free. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) Proceedings of MoDELS/UML 2005, 9th International Conference. pp. 155–169. LNCS 4199, Springer, Genova, Italy (2006)
30. Reenskaug, T., Coplien, J.O.: The DCI architecture: A new vision of object-oriented programming. `http://www.artima.com/articles/dci_vision.html` (3 2009)
31. Reenskaug, T., Wold, P., Lehne, O.A.: Working With Objects: The OOram Software Engineering Method. Prentice Hall (1996)
32. Sánchez, P., Fuentes, L., Jackson, A., Clarke, S.: Aspects at the right time. Transactions on Aspect-Oriented Software Development IV, 54–113 (2007)
33. Steimann, F.: On the representation of roles in object-oriented and conceptual modelling. Data & Knowledge Engineering 35, 83–106 (Oct 2000)
34. Steimann, F.: Domain models are aspect free. In: Proceedings of ACM/IEEE 8th International Conference On Model Driven Engineering Languages And Systems, MoDELS 2005. pp. 171–185. LNCS 3713, Springer, Montego Bay, Jamaica (2005)
35. Vranić, V., Bebjak, M., Menkyna, R., Dolog, P.: Developing applications with aspect-oriented change realization. In: Proceedings of 3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2008, Revised Selected Papers. LNCS 4980, Springer, Brno, Czech Republic (Oct 2011)
36. Vranić, V., Michalco, P.: Are themes and use cases the same? Information Sciences and Technologies, Bulletin of the ACM Slovakia, Special Section on Early Aspects at AOSD 2010 2(1), 66–71 (2010)

Valentino Vranić is an associate professor of software engineering at the Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies of the Slovak University of Technology in Bratislava, Slovakia. He explores different aspects of software development. In particular, he is interested in preserving intent comprehensibility in code and models using advanced modularization, as well as in effective agile and lean organization of software development.

Milan Laslop received an MSc. in software engineering from the Faculty of Informatics and Information Technologies of the Slovak University of Technology in Bratislava, Slovakia. Currently, he works as a software engineer at ESET in Bratislava, Slovakia.

Received: August 31, 2014; Accepted: December 7, 2015.