Classifying and documenting aspect interactions

Frans Sanen, Eddy Truyen, Wouter Joosen
Distrinet Research Group
Department of Computer Science
K.U.Leuven
Celestynenlaan 200A,
B-3001 Heverlee - Belgium
Tel. (+32) (0) 16 32 76 02
frans.sanen@cs.kuleuven.be

Andrew Jackson, Andronikos Nedos, Siobhán Clarke
Distributed Systems Group,
Trinity College Dublin
Dublin 2 - Ireland,
Tel (+353) (0) 16081543
anjackso@cs.tcd.ie

Neil Loughran, Awais Rashid
Computing Department
Infolab 21,
Lancaster University
(+44) (0) 1524 510503
loughran@comp.lancs.ac.uk

ABSTRACT
In this position paper, we propose an approach for classifying and documenting aspect interactions. Making aspect interactions explicit will result in an important form of knowledge that can be shared or used over the course of system evolution.

Keywords
Aspect-Oriented Software Development, Interaction

1. INTRODUCTION
Aspects enable isolating and modularizing crosscutting concerns. In the literature, aspects that are orthogonal to one another have often been used to illustrate the benefits of Aspect-Oriented Software Development (AOSD) [8]. However, are all aspects truly orthogonal in reality? It is a simple question with an equally simple answer. Beyond some trivial lower-level aspects, there will be interdependencies between the more interesting aspects, resulting in aspect interactions.

Our motivation is based on the relevance of the problem. Aspect interactions are both technology and domain independent. The domain independence can be motivated by the example interactions that are known for numerous application domains, such as telecommunications, email, thermo control, policy-based, multimedia, middleware and other systems [2, 3, 4, 6, 11, 13, 15, 16]. Due to this technology and domain independence, solutions for aspect interactions will be generally applicable. As a consequence, aspect interactions can directly endanger the integrity, availability and reliability of their enclosing infrastructure. Future expectations aren't good either, because, in our opinion, the amount of aspect interactions will keep increasing. Since software development is subject to reduced time-to-market cycles, an aspect-oriented system will need to cope with evolution beyond delivery and unanticipated changes during the maintenance phase. Additionally, more and more aspects will be developed by third parties, independently of the existing platform. Garlan et al. [10] argument that software elements developed independently of each other are the main reason causing interactions because they make implicit or explicit assumptions that are only valid if the software element is operating in isolation. Applying their idea to AOSD gives us aspects, which make implicit or explicit assumptions that are only valid if the aspect is the only aspect composed with the base system, as the main reason causing interactions. Unfortunately, current systems cannot guarantee correctness when aspects are not tightly integrated with the base system.

There exists a large set of publications on interactions, especially in the telecommunications domain [4, 13]. However, interactions also arise in complex systems such as middleware and product lines [2, 15]. In the aforementioned systems, there is a common agreement and wide acknowledgement of the necessity of handling interactions. First of all, it is important to realize that not all interactions are harmful. Both positive and negative aspect interactions exist. Therefore, we propose a classification of aspect interactions in which this distinction is reflected. The main benefit from this classification is that by having different types of interactions common patterns of interaction and their response types may arise. Once we have a substantial list of interactions, analysis of the examples for a specific type of interactions may reveal these common patterns. Secondly, to the best of our knowledge, a means for explicitly documenting aspect interactions is lacking. Therefore, we propose an approach for documenting aspect interactions, and, hence, making them explicit. This will result in an important form of knowledge that can be shared or used over the course of system evolution. Aspect interactions then can be addressed if they are negative or kept as part of system documentation if they are positive.

Obviously, this form of knowledge about interactions can be useful at different stages of the software life cycle. For instance, a middleware platform can be developed to be able to interpret implementation-oriented interactions automatically whereas an architectural design tool can exploit interactions exposed by requirements-level composition and analysis. Especially the latter case of an architectural blueprint capturing design know-how that resolves certain axiomatic interactions – interactions that are invariant – is the long term result we aim for.

The rest of this paper is structured as follows. In Section 2 we present the example application we use to give examples of aspect interactions. We propose a classification of aspect interactions in Section 3. Our proposed approach to document aspect interactions explicitly is elaborated and illustrated in Section 4. Section 5 discusses our work. Some related work is covered in Section 6. Finally, this position paper is concluded in Section 7.

2. EXAMPLE APPLICATION
The example application we use to give examples of aspect interactions in our classification is an online auction system that allows people to negotiate over the buying and selling of goods in the form of English-style auctions. The system only allows for enrolled users to log into the system to start a session in which they can buy, sell or browse through auctions available. When one wants to follow an auction, one must first join the auction. Once

one has joined an auction, one may make a bid. One only can bid before an auction closes. After it is closed, the system calculates the highest bid and checks if it meets the reserve price issued by the seller. Finally, the relevant parties are informed of the outcome of the auction.

In this application context, we limit our scope to three concerns[1]: persistence, security and context-awareness. Based on discussions with experts from all three domains, we came up with an initial breakdown for these concerns into sub-concerns in order to make the interaction documentations as concrete as possible. An important criterion for these breakdowns was the orthogonality of the sub-concerns to structure further analyses more easily. Security encloses authentication, authorization, non-repudiation, integrity, confidentiality and auditing. Persistence is broken down into state encoding, state change detection, state-access, transactions, caching and logging. Our context-awareness concern breakdown distinguishes between context monitoring, inference and action. We start from the assumption that all these sub-concerns can be realized through aspects.

3. CLASSIFICATION

Aspects can interact in multiple ways. In what follows, we distinguish between four different types of aspect interactions: mutual exclusion, dependency, reinforcement and conflict.

- *Mutual exclusion* encapsulates the interaction of mutual exclusiveness. For example, if there are two aspects that implement similar policies, or algorithms, the situation can arise where only one such aspect must be used. No mediation is possible because the aspects are not complementary: only one of them can be used, the other cannot. With respect to the auction example this might simply be the choice of a scheme for caching particular records in order to increase performance. If two mutually exclusive schemes were weaved, results will be unpredictable or undesirable. Another example might be a specific product configuration using persistence that included both relational and object mapping schemes, which could result in data duplication or wrong encoding in the target database.
- *Dependency* covers the situation where one aspect explicitly needs another aspect and hence depends on it. A dependency does not result in a problem or erroneous situation as long as the aspect on which another one depends is ensured to be present and not changed. To illustrate this situation, two simple dependencies are the following: authorization depends on authentication and context actuation depends on context monitoring. Without the latter, the former cannot perform correctly.
- *Reinforcement* arises when an aspect influences the correct working of another aspect positively and hence reinforces it. There can be no doubt that this type of interaction is a positive one. When an aspect reinforces another aspect, extended functionalities become possible and extra support is offered. As an example, the monitoring of an auction user's location (context-monitoring) allows for an extended authorization policy. For instance, it is possible to realize that only users within the country in which an auction is taking place are allowed to make a bid.
- *Conflict* captures the situation of semantical interference: one aspect that works correct in isolation does not work correctly anymore when it is composed with other aspects. Hence, an aspect influences the correct working of another aspect negatively. Typically, a conflict can be solved by mediation because the aspects –in a sense- are complementary. To illustrate this situation, non-repudiation and state encoding are in conflict because one does not want to persist repudiated data.

4. DOCUMENTING ASPECT INTERACTIONS

4.1 Template

We propose a template with clearly defined semantics for describing interactions unambiguously. This template consists of a record structure consisting of a number of attribute-value pairs and is explained below.

We have found that, when documenting aspect interactions, it is important to explicitly state the relevant conditions that hold when an interaction occurs. Obviously, these conditions evaluate over the context of the interaction. In our template, these conditions are called explaining predicates.

An aspect interaction description consists of the following attributes:

- *Name*, containing the name of the interaction.
- *Aspects involved,* containing the aspects involved in the interaction. Involved aspects typically are at the level of sub-concerns.
- *Type*, containing the type of interaction. We distinguish between four different types of interaction: mutual exclusion, dependency, reinforcement and conflict. These were discussed in Section 3.
- *Example,* containing an informal example of the interaction.
- *Explaining predicates,* containing the predicates that can be used to explain the interaction. Predicates typically express conditions that evaluate over a number of parameters representing the relevant context information of the interaction. Each explaining predicate is again described through attribute-value pairs.
  - *Name,* containing the name of the explaining predicate.
  - *Aspect,* containing the aspect the explaining predicate is referring to.
  - *Definition,* containing the definition indicating in which states and/or under which conditions the explaining predicate holds.
  - *Parameters,* containing the parameters that are relevant for evaluating the explaining predicate.
- *Description,* containing an informal description of the interaction in terms of the explaining predicates.
- *Time of response,* containing the appropriate time in the software lifecycle to respond to the interaction. Software lifecycle stages we consider to be relevant are the following: requirements, architecture, design, implementation, middleware, deployment.
- *Type of response,* containing an informal description of the mechanism, action and/or structure that uses the knowledge about the interaction to solve it (in case of a conflict or mutual exclusion) or instantiate it (in case of a dependency or reinforcement).

[1] These three concerns were already studied as a part of [21].

4.2 Illustrated

We illustrate the applicability of our template in documenting aspect interactions with an example taken from the online auction system as described in Section 2. Users can participate in an auction through their handheld device. As mentioned in the application description, each buyer and seller will have an associated session. In order to ensure integrity of each bid, a symmetric session key is part of each user session. For obvious reasons, this session key only has a limited lifetime reducing the chance that the integrity of a bid can be broken. Each time the session key expires, a new one has to be generated, which is very computationally intensive. When the handheld device power is low, a clear aspect interaction arises. The integrity security sub-concern is in conflict with the device power context, in which one wants to avoid intensive computations. The example described within our template is shown in Table 1.

Table 1. Example aspect interaction description.

Name	Extend session key lifetime if device power is low
Aspects involved	Integrity, Context monitoring
Type	Conflict
Example	There shouldn't be generated a new symmetric session key when the power of an auction user his/her handheld device is low.
Explaining predicates	

Name	SessionKeyAboutToExpire
Aspect	Security/Integrity
Parameters	Joinpoint
Definition	SessionKeyAboutToExpire(Joinpoint jp) holds when the current session key is about to expire at join point jp.

Name	DevicePowerLow
Aspect	Context-awareness/Context monitoring
Parameters	Joinpoint
Definition	DevicePowerLow(Joinpoint jp) holds when the device power is low at join point jp.

Description	∀ Joinpoint jp: DevicePowerLow(jp) requires !SessionKeyAboutToExpire(jp)
Time of response	Architecture, design, middleware
Type of response	When the power of one's handheld device is low, the lifetime of the current symmetric session key should be automatically extended instead of creating a new key.

5. ONGOING AND FUTURE WORK

This approach is being taken within a larger investigation of AOSD applications[2] carried out in the context of AOSD-Europe [1]. We have begun to investigate aspect interactions within the context of the online auction system and the scope of three extra-functional concerns: persistence, security and context-awareness. Currently, we are manually documenting a list of aspect interactions between these aspects in the example auction application. Our previous example illustrating the template is one of these.

Based on this first experiment, we will evaluate the proposed documenting approach and extend it where needed. Extra attribute-value pairs may be needed. For instance, we do not consider instances of aspects yet. For now, we assumed there is only one global instance of each aspect. Moreover, we believe that other (sub)types of interaction may exist and will need to be incorporated. Our end goal is an exhaustive and non-overlapping classification of aspect interactions.

We realize that, because of the limited scope of our experiment, this merely is a first step towards our end goals. Nevertheless, we are persuaded of the relevance of finding a good approach to document aspect interactions explicitly and unambiguously. By means of documenting aspect interactions and, hence, make them more explicit, it will be possible to share this crucial development and execution knowledge. This results in an important form of help for other members of the team or over the course of evolution, when even a different team may be in place. Classifying and documenting aspect interactions enables us to address them in case of a conflict or a mutual exclusion or have them as part of the system documentation in case of a reinforcement or a dependency.

6. RELATED WORK

Interactions are a widely known problem, especially in the telecommunications domain. The relevance of interactions in the broader context of a middleware platform and its proliferated common services has been discussed in [2, 15]. A number of solutions have been proposed to deal with conflicting situations, such as for example in [17], where the authors propose the notion of a compositional intersection to enable the identification of suitable sets of concerns that can then be used in an early trade-off analysis. Some first results based on an analysis of a set of aspects are discussed by Douence et al. [7] and Rinard et al. [19].

We are convinced that our approach for classifying and explicitly documenting interactions is complementary to this body of existing work. To the best of our knowledge, there only exist a few approaches that are similar to our proposed approach and focus on describing the interactions themselves. In [18], Pawlak et al. propose CompAr, a language that allows programmers to abstractly define an execution domain, advice codes and their often implicit execution constraints. Their language enables the automatic detection and solving of aspect-composition issues (aspect interactions) of around advices. Major contribution of the work in [18] is the high level of abstraction the language offers to specify very generic aspect definitions. Batory et al. [14] propose

[2] More information about this larger investigation of AOSD applications can be found on the AOSD-Europe web portal (http://www.aosd-europe.net) within the applications research lab area.

an algebraic theory for modeling interactions in feature-oriented designs. In their theory, feature interactions are modeled as derivatives. The approach taken in [14] is very similar to the Distributed Feature Composition of Zave et al. [12].

7. CONCLUSION

In this short paper we have demonstrated our approach for classifying and documenting aspect interactions. While the research presented here is still in the preliminary phases, we believe that the direction taken offers a fresh perspective that is complementary to other research already undertaken in the field of AOSD. Indeed, we already envisage that the work could form the beginnings of what will be a catalogue of *interaction patterns* in a similar vein to that of design patterns [9]. A future document [20] will express this in more detail.

So far we have only discussed *production aspects* (i.e. aspects that are included in the final application). Of course, in a typical development scenario one could imagine many different kinds of *development aspects* (i.e. aspects that are used in the activities of testing, profiling, tracing, where the code is not included in the final release) causing interactions with our core concerns. Additionally different *versions* of concerns, which may exist in the software product line [5] context, where concerns are customized to differing requirements, can also further exacerbate the problem of interactions.

Interactions across many aspects add to our conviction that there is a need for the documentation, detection, modularization and, ultimately, the resolution of aspect interactions.

8. ACKNOWLEDGMENTS

This work is supported by European Commission grant IST-2-004349: European Network of Excellence on Aspect-Oriented Software Development (AOSD-Europe), 2004-2008.

9. REFERENCES

[1] AOSD-Europe. European Network of Excellence on Aspect-Oriented Software Development. European Commission grant IST-2-004349.

[2] L. Blair, G. Blair, and J. Pang. Feature interaction outside a telecom domain. *Workshop on Feature Interaction in Composed Systems*, 2001.

[3] L. Blair, and J. Pang. Feature interactions - Life beyond traditional telephony. *FIW 2000*, p. 83-93.

[4] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec. Feature interaction: A critical review and considered forecast. *Computer Networks: The International Journal of Computer and Telecommunications Networking archive*, Vol. 41 , Issue 1, p. 115-141, January 2003.

[5] P. Clements and L. Northrop, "Software product lines - Practices and patterns," Addison Wesley, 2002.

[6] J. A. Diaz Pace, F. Trilnik, and M. R. Campo. How to handle interacting concerns? *Workshop on Advanced for Separation of Concerns in OO Systems*, OOPSLA 2000, Minneapolis, USA.

[7] R. Douence, P. Fradet, and M. Sudholt. Composition, reuse and interaction analysis of stateful aspects, *International Conference on Aspect-Oriented Software Development (AOSD04)*, 2004.

[8] R. E. Filman, T. Elrad, S. Clarke, and M. Aksit (Eds.). Aspect-oriented software development. Addison-Wesley, 2005.

[9] E. Gamma, et al. Design patterns: Elements of reusable object-oriented software. Addison-Wesley, Reading, 1995.

[10] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch or why it's hard to build systems out of existing parts. *Proceedings of the 17th International Conference on Software Engineering*, Seattle, Washington, April 1995

[11] R. J. Hall. Feature interactions in electronic mail. *Proceedings of the 6th International Workshop on Feature Interactions in Telecommunications and Software Systems*, IOS Press, 2000.

[12] M. Jackson, and P. Zave. Distributed feature composition: A virtual architecture for telecommunications services. *IEEE Transactions on Software Engineering* XXIV(10):831-847, October 1998.

[13] D. O. Keck, and P. J. Kuehn. The teature and service interaction problem in telecommunications systems: A survey. *IEEE Transactions on Software Engineering*, Vol. 24, No. 10, October 1998.

[14] J. Liu, D. Batory, and S. Nedunuri. Modeling interactions in feature oriented systems. *International Conference on Feature Interactions (ICFI)*, June 2005.

[15] X. Liu, G. Huang, W. Zhang, and H. Mei. Feature interaction problems in middleware services. *International Conference on Feature Interactions (ICFI)*, June 2005.

[16] E. Lupu, and M. Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, Vol. 25 , Issue 6, p. 852-869, November 1999.

[17] A. Moreira, A. Rashid, J. Araujo. Multi-dimensional separation of concerns in requirements engineering. *International Conference on Requirements Engineering*, Paris, France, 2005.

[18] R. Pawlak, L. Duchien, and L. Seinturier. CompAr: Ensuring safe around advice composition. *7th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS05)*, Athens, Greece, June 2005.

[19] M. Rinard, A. Salcianu, and S. Bugrara. A classification system and analysis for AO programs. *Proceedings of the Twelfth International Symposium on the Foundations of Software Engineering*, Newport Beach, CA, November 2004.

[20] F. Sanen, E. Truyen, W. Joosen, N. Loughran, A. Rashid, A. Jackson, A. Nedos, S. Clarke. Study on interaction issues. AOSD-Europe Deliverable 44. March 2006.

[21] F. Sanen, E. Truyen, B. De Win, W. Joosen, N. Boucke, T.Holvoet, N. Loughran, A. Rashid, R. Chitchyan, N. Leidenfrost, J. Fabry, N. Cacho, A. Garcia, A. Jackson, N. Hatton, J. Munnelly, S. Fritsch, S. Clarke, M. Amor, L. Fuentes, L. Fuentes, and C. Canal. A Domain Analysis Of Key Concerns – Known And New Candidates. AOSD-Europe Deliverable 45. March 2006.