Expressing Aspectual Interactions in Requirements Engineering

Experiences in the Slot Machine Domain

Arturo Zambrano
LIFIA, Facultad de Informática, Universidad Nacional de La Plata
50 y 115 1er Piso
La Plata, Argentina
arturo@lifia.info.unlp.edu.ar

Johan Fabry
PLEIAD laboratory, Computer Science Department (DCC), University of Chile
Blanco Encalada 2120,
Santiago, Chile
jfabry@dcc.uchile.cl

Guillermo Jacobson
LIFIA, Facultad de Informática, Universidad Nacional de La Plata
50 y 115 1er Piso
La Plata, Argentina
gaj@lifia.info.unlp.edu.ar

Silvia Gordillo
LIFIA, Facultad de Informática, Universidad Nacional de La Plata
CIC Pcia. Buenos Aires
50 y 115 1er Piso
La Plata, Argentina
gordillo@lifia.info.unlp.edu.ar

ABSTRACT

Aspect Oriented Requirements Engineering (AORE) provides support for modularizing crosscutting requirements. In the context of an industrial project in the domain of Slot Machines we needed to perform AORE, with a special emphasis on dependencies and interactions among concerns. We were however unable to find any report of large-scale industrial applications of AORE approaches that treat dependencies and interactions. We therefore evaluated two AORE approaches: Theme/Doc and MDSOCRE, to establish their applicability in our setting. In this paper we report on our experience, showing successful uses of both approaches as well as where they fall short. We furthermore propose possible enhancements for both approaches, to address these limitations.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements / Specifications—*Methodologies, Tools*

Keywords

aspect oriented requirement engineering, aspect dependencies and interactions

SAC'10 March 22-26, 2010, Sierre, Switzerland.

1. INTRODUCTION

Aspect-Oriented Requirements Engineering (AORE) addresses the requirements engineering problem that some requirements are hard, if not impossible, to isolate into separate modules. Also known as Early Aspects, AORE performs first-class modelling of these crosscutting concerns as aspects, identifying and characterising their influence on other requirements in the system [15, 17]. These models enable to better identify and manage requirements conflicts, irrespective of the crosscutting nature of the requirement. Ideally, the result of this phase is to have a consistent model of the system early in the software development life-cycle.

In the context of an industrial project we are currently re-implementing the software that runs on the casino gambling device best known as a slot machine. Our previous experience with this software has taught us that there is a significant amount of crosscutting concerns in these applications. Furthermore these concerns depend on and interact with each other as well as with the modularised concerns. We therefore have opted to use Aspect-Oriented Software Development in this implementation, taking special care of dependencies and interactions between the different aspects and modules.

Being aware of the critical importance of interactions in this domain, we have focused early in the development cycle on interaction modeling. The objective of this step is documenting as many interactions as possible so that this information can be used later in the design and implementation phases. The result of the modeling process should be a consistent model of the requirements. To accomplish this objective, it is necessary to be able to rely on expressive mechanisms in the selected modeling techniques for this phase.

To the best of our knowledge there are however no pub-

lications detailing experiences with AORE approaches and their interaction support in a large-scale industrial case. We therefore opted to perform an in-depth study of two approaches to evaluate their applicability in the slot machine domain. This paper presents our results.

Applying existing aspect oriented software engineering tools and approaches to real world cases is necessary to bring maturity to them and finally gain the acceptance of the rest of the software engineering community. We believe this paper contributes to this field.

We elected to perform requirements engineering using both the Theme/Doc approach [3] and Multidimensional Separation of Concerns for Requirements Engineering (MDSOCRE) [15], focusing on how these allow us to express and document aspectual dependencies and interactions. The choice of these two approaches was mainly based on our perception of their maturity and of their acceptance in the AORE community, the latter as reflected by publication record. Due to lack of time, we were unable to perform the same experiment on other approaches such as AORA [6], AOSD/UC [14], or AOREC [13].

Both of the approaches we evaluated enable us to express the requirements, but neither of them was entirely satisfactory. Theme/Doc lacks support for the kind of interactions we want to model – e.g. conflicts between aspects – and needs a finer granularity for expressing crosscutting relationships. MDSOCRE lacks explicit support for expressing interactions between concerns when they do *not* cross-cut each other. In addition to describing these drawbacks in more details, we also provide suggestions on how they may be addressed in these tools, for example adding new kind of relationships to Theme/Doc for documenting interactions, or adding new meta-information about concerns for MDSOCRE. We hope that this experience report is of use to the AORE community, indicating possible enhancements of the tools and methodologies. This experience is also useful for those who want to perform AORE in domains where interactions are present, to know which support they will get from the evaluated tools.

This paper is organised as follows: Sect. 2 is a brief introduction to the slot machine domain and the specificity of requirements in this domain. In Sect. 3 we present our decomposition of the application in different concerns, along with a textual description of requirements and interactions. Our first modelling effort is described in Sect. 4 that presents our results using Theme/Doc. Section 5 repeats the experiment, using MDSOCRE. We present related work in Sect. 6 and Sect. 7 concludes.

2. SLOT MACHINE DOMAIN

A slot machine (SM for short) is a gambling device. It has five reels which spin when a *play* button is pressed. An SM includes some means for entering money, which is mapped to credits. The player bets an amount of credits on each play, the SM randomly selects the displayed symbol for each reel, and pays the corresponding prize, if any. Credits can be extracted (called *cashout*) by different mechanisms such as coins, tickets or electronic transfers.

The SM game concept is developed by the game designers and its implementation must obey a set of regulations that control both hardware and software. As a game concept can vary from SM to SM, in this paper we only focus on the legal regulations, and we furthermore restrict ourselves to regulations for software. These can be divided in three main groups:

Government Regulations: Government regulations cover a broad spectrum of characteristics of gambling devices: payout, randomness, connectivity, shared prizes, etc. One example of these are the Nevada Regulations [16].

Standards: To ensure proper behaviour of SMs, there are certification institutes that perform severals tests and quality checks on the SMs. The expected behaviour of an SM is defined in documents called standards, for example the GLI standard [11].

Technical Specifications: Some requirements are related to the SM's connectivity with *reporting systems* (RS) and the underlying communication protocol. This is the case, for example, of the G2S [12] (Game to Server) protocol, an open standard for communication of SM with a backend.

Requirements for the SM domain, are therefore defined in different documents (regulations, standards, protocol specifications) written by different stakeholders, with diverse interests and backgrounds. This results in a large set of documents using multiple terms for describing the same object, action or event (we count approximately 150 pages of requirement documents). Furthermore, in some cases it is necessary to complement and normalise different sources referring to the same topic. For instance, consider the case of *Error Conditions*, which are treated by both Nevada regulations [16] and the GLI standard [11]; some of the conditions specified by the regulations match, but others are defined by just one of them.

Lastly, an important characteristic of communication protocol requirements (e.g. in G2S) is that they are divided in topics and that not all the topics are required for certain deployments. As a result, part of the communication functionality is optional, which has an impact on requirements modeling.

3. CONCERNS

In AORE, concerns refer to a coherent set of requirements that allude to a property or feature that the system must provide [7]. Based on our experience in the domain and the set of legal requirements that apply, we organise the requirements in the following concerns:

Game: This is the basic logic of a gambling device. The user can enter credits into the machine, and then play. The output is determined randomly and when the player wins, he is awarded an amount of credits.

Metering: This refers to a set of counters that are used to audit the activity of the game. For instance, there are meters that count the number of plays, the total amount bet, total won, error condition occurrences, etc.

Program Interruption and Resumption: Requirements in this concern determine how the machine should behave after a power outage, specifying which data and state need to be recovered.

Game Recall: This refers to the information that must be available about the current and previous plays, in order to solve any dispute with players.

Error Conditions: Under certain circumstances, the game should detect error conditions and behave accordingly. This concern defines what are considered error conditions and how the game must react to them.

Communications: The SM is connected to a reporting

system (RS) in the casino. This concern defines the kinds of data, the format and when data must be exchanged between the SM and RS. Several communication protocols exist, each with their own specification that states what data needs to be persistent, which meters are necessary, etc. Concretely, we will either use G2S or a proprietary protocol, and therefore in the remainder of the text refer to the type of protocol instead of to 'communications'.

Demo: The demo concern contains the requirements specifying how the game behaves in this mode. Playing the game in demo mode makes it is possible to test how hardware and software works, simulating events such as entering money or winning a prize[1]. Note that any meter or data changed due to operation in demo mode should not be persisted or reported, as it is simulated behaviour.

As the domain is complex, there may be other possible concern decompositions. We choose this one because, based on our experience, it properly modularises the different required features of slot machines and show the interactions that are at the core of this evaluation.

We expect some of the selected concerns to become components and others aspects. Different instantiations of the SM software will include different components or aspects to comply with the regulations of each scenario.

3.1 Selected Requirements

Due to the large number of requirements of our case, we only show here a small subset of requirements that illustrates 3 important types of interactions discussed later: conflict, dependency and reinforcement.

The *Meters* concern contains multiple requirements, some examples are: "Meters shall at all times indicate all credits or cash available for the player to wager or cashout", "Meters should be updated upon occurrence of any event that must be counted, including: play, cashout, bill in, coin in, handpay, etc.", "Meters must count bills/tickets/coins inserted, cashout amount, etc.". In the case of *Demo*, we have as the principal requirement: "Plays and other actions such as cashout, handpays, etc. performed in this mode must not be counted for audits". For the *Communication Protocol* concern we have: "Information regarding SM activity must be sent to the RS", "Meters reported by the protocol includes number of plays, bills inserted by denomination, error conditions, etc" as most important.

A more detailed list of the specific requirements of our experiment, selected from the official requirements documents [11, 12, 16], is available from our web site: `http://pleiad.cl/research/adi`.

3.2 Interactions

Aspectual interactions have been studied by several authors. We based our investigation, and therefore this discussion on the AOSD-Europe technical report that gives an overview on aspect interactions [20]. In this report, the authors classify aspect interactions into dependency, conflict, mutex and reinforcement. For example, in the SM domain, there is a conflict between the *Demo* and *Meters* concerns, since *Meters* works correctly without *Demo*, but if *Demo* mode is active, activity in the machine must not be counted by *Meters*. An example of mutex is in the communication protocols: it is forbidden to have two protocols providing the same functionality at the same time. A dependency example is the relationship between *Communications* and *Meters*; the protocol needs to communicate the status of the SM, which is in part represented by the meters. Finally, a reinforcement is a positive interaction, for instance between *Error Conditions* and *Communications*. The existence of error condition detection enables communication protocols to provide "extra" functionality, in this case real time error condition reporting.

Understanding how concerns interact with each other is key information that needs to be passed to designers and programmers. For example, in the case of a dependency the dependent concern will be affected by design decisions on the other. On the other hand, if there is a mutex relationship, architectural mechanisms should be provided to ensure that both aspects will not be active at the same time.

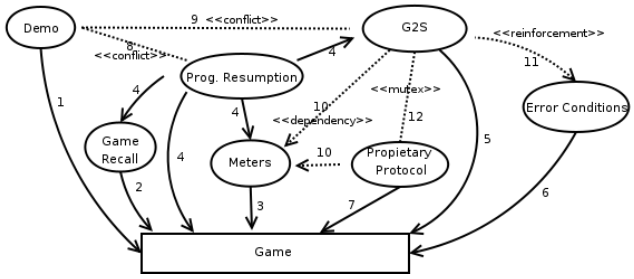

Figure 1: Concern interactions. Regular arrows indicate crosscutting, dashed arrows indicate interactions between concerns, tagged with UML-like stereotypes.

Considering the concern division and the associated requirements, we have deduced the relationships between different concerns and identified their interactions, as shown in Fig. 1. More in details, the **crosscutting** relationships are as follows:

1. *Demo* to *Game*: The demo requirements affect many of the definitions of the original requirements of *Game*, in order to alter the *Game*'s behaviour for testing purposes.

2. *Game Recall* to *Game*: *Game Recall* requirements affect many aspects of the *Game*'s behaviour, its requirements call for recording pieces of information regarding game play.

3. *Meters* to *Game*: *Meters* count activities of many functions defined in *Game*'s requirements, for instance: game play, bill in, cashout, etc.

4. *Program resumption* to *Game*, *Game Recall*, *G2S* and *Meters*: *Program resumption* is analogous to persistence. It crosscuts all the places where important data, which needs to be restored, is changed.

5. *G2S* to *Game*: this concern cut across many *Game*'s requirements, since several events in Game need to be reported, monitored and communicated to reporting system.

6. *Error Conditions* to *Game*: The behaviour associated to error conditions need to be woven into the game

[1]SMs are delivered to casinos without the demo feature installed, obviously.

behaviour. Requirements in *Game* that could raise an error condition vary: from a bill inserted to a door opened.

7. *Proprietary Protocol* to *Game*: This refers to another protocol used for the same purpose as G2S, that is to monitor the game's behaviour.

The different **interactions** are the following:

8. Conflict between *Demo* and *Program Resumption*: The demo mode fires *fake* events that must not be counted nor restored after program interruption.

9. Conflict between *Demo* and *G2S*: This means both concerns cannot be active at the same time, because demo fires fake events that must not be reported to the RS.

10. Dependency of *G2S* and *Prop. Protocol* on *Meters*: Some data reported to the RS is stored or can be derived from meters. So, communication protocols needs the meters to be up to date in order to accomplish its purpose.

11. Reinforcement of *G2S* with *Error Conditions*: As we mentioned in Sect. 2, some parts of the G2S protocol are not mandatory for specific instances. When error conditions are tracked in the game, additional behaviour is made available in G2S, such as real time event reporting.

12. Mutex between *G2S* and *Proprietary Protocol*: There is overlapping functionality defined in the requirements of both protocols. For example, both of them are used keep the time in sync between the SM and the RS. Having both protocols active, with RSs out of sync, would render the time of the SM inconsistent. Therefore they cannot be active at the same time.

The interactions of these concerns need to be taken into account at design time. Therefore this must be clearly reflected in the model resulting from the Requirements Engineering phase. We evaluated two AORE approaches to establish whether they provide the necessary expressiveness for our needs. In the next sections we discuss Theme/Doc [3] and Multidimensional Separation of Concerns for Requirements Engineering [15], showing if and how they make it possible to express and document aspectual interactions.

4. EVALUATION OF THEME/DOC

Theme/Doc is an AORE methodology that, apart from being mature and accepted in the AORE community, is part of a more comprehensive approach called Theme [3, 10], which also treats aspectual design (Theme/UML). We selected Theme/Doc because it explicitly supports passing information from requirements analysis, which could include interactions, to the design phase.

4.1 Brief overview of Theme/Doc

Theme/Doc [2] is the requirement analysis part of the Theme approach [3, 10]. In Theme/Doc, requirements are organised into concerns, called *themes*. Themes can be defined through an initial set of domain specific actions or concepts, others may be recurring typical concerns: persistence, logging, etc.

In Theme/Doc a requirement is attached to a theme if the name of the theme appears in the requirement. In other words, Theme/Doc relies on the name-based analysis of actions in requirements to relate them to themes. In our study we did not strictly follow this rule. Instead we use the concerns we identified in Sect. 3 as themes. We will detail our motivation for this in Sect. 4.3.

Ideally, each requirement should belong to one theme, but chances are that some of them are shared among themes, i.e. crosscutting. In Theme/Doc, a shared requirement is considered crosscutting if the following four conditions are satisfied [10]:

1. The requirement cannot be split in order to avoid tangling.
2. One of the themes dominates the requirement: it has a stronger belonging relationship with one of the themes.
3. The dominant theme is triggered by events in the base theme: the behaviour described by the crosscutting theme is fired as a result of the execution of some behaviour from the base theme.
4. The triggered theme is externally fired in multiple situations: the crosscutting behaviour must be executed in several cases, not just one.

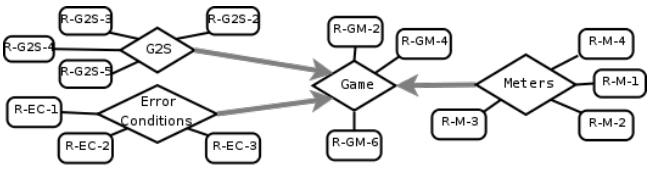

Figure 2: *Game*, *Meters*, *G2S* and *Error Conditions* concerns expressed using Theme/Doc notation.

An important feature of Theme/Doc is its visual support through diagrams. In Theme/Doc views, requirements are represented by rounded boxes, and they are organised around themes, which are depicted by diamonds. When a crosscutting theme exists, a gray arrow is drawn from the theme that crosscuts (*i.e. the aspect*) to the theme that is being cut across (*i.e. the base*). Consider for example Fig. 2, where *Game*, *Meters*, *Error Conditions* and *G2S* concerns are represented along with their requirements and crosscutting relationships.

4.2 Successful Uses of Theme/Doc

As shown in Fig. 2, the graphical approach of Theme/Doc makes it easy to read the relationships between requirements and themes. Each theme can be easily identified along with its associated requirements. The four steps to check for crosscutting helped us to confirm which are the crosscutting concerns. In the resulting diagrams, the crosscutting relationships are clear, enabling us to easily identify which concern is playing the base and/or the aspectual role.

Figure 3 shows crosscutting among the themes presented in Sect. 3.2. For clarity, we just present the crosscutting relationships between the themes and without including the requirements[2].

[2]Diagrams showing all the above concerns with their more significant requirements can be found on our website: `http://pleiad.cl/research/adi`.

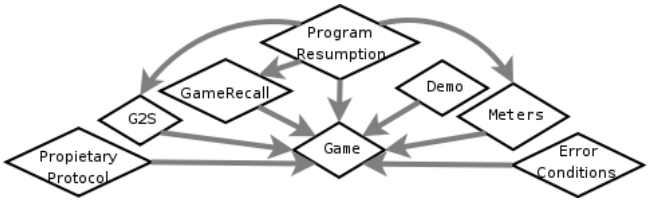

Figure 3: Crosscutting relationships between themes using Theme/Doc graphical notation.

For the application of the approach we followed the instructions from the published work [10, 3]. We needed to generate some views manually as the Theme/Doc tool was not available at the time that we performed our experiments.

4.3 Limitations of Theme/Doc

In our evaluation, we encountered the following limitations of Theme/Doc.

Granularity: As explained before, gray arrows denote crosscutting. As each concern potentially contains many requirements, it is difficult to discern which specific requirement of the crosscutting theme affects which requirements on the base theme. Consider for example Fig. 2 and the crosscutting relationship between *Meters* and *Game*; here it is not possible to know which requirement in *Meters* is crosscutting. Furthermore, it is not possible to know which specific requirements in *Game* are affected as the result of the crosscutting. Where possible, it is desirable to pass that information to the design phase, so that base and aspectual components can be properly designed. In fact, this information is available during the analysis phase –identification of crosscutting themes– of Theme/Doc, but it is not made explicit.

Expressing Interactions: In Fig. 1 we show different examples of interactions between aspectual concerns for requirements. If we consider Fig. 2 we can however see that the interactions explained in Sect. 3.2 are missing. This is because Theme/Doc lacks support for expressing interactions. For instance, missing in Fig. 2 is a dependency of *G2S* on *Meters*. This information is however crucial: Multiple perspectives of a system (*themes* in this case) need to be combined to form a system [21]. We require the dependency information to select a sound set of themes for a system. For example, it is not possible to build an SM with *G2S* support but lacking *Meters*. This is because *G2S* requires the existence of *Meters* to provide its own functionality. The same happens with *conflicts*, for instance, between *Demo* and *Meters*. It is critical to know that architectural or design mechanisms need to be included to avoid the activation of both concerns at the same time. Developing the system without this information would entail costly fixes in the future, when the interaction is encountered. The reinforcement from *Error Conditions* to *G2S* is also missing. Documenting it signals that an optional part of G2S is active when *Error Conditions* are available.

Adaptability to our case study requirements: A useful requirements specification must be complete, unambiguous, verifiable, consistent, modifiable and traceable [1]. Under these assumptions Theme/Doc should work smoothly. Unfortunately, in our particular case there is no single requirements specification unifying all the sources and we are faced with significant ambiguity. The variety of sources results not only in synonyms being used in different documents. There are complete key ideas, concepts or interactions that are expressed using different vocabulary and style. Although we might consider our case as being exceptional, it is worthwhile to examine the impact this has on Theme/Doc.

The ambiguity we face affects the mechanism proposed in Theme/Doc to assign requirements to themes, and to identify potential crosscutting themes. For example, consider the case of attaching requirements to themes, where it is necessary to look for a theme's name in the requirements. In our case, sometimes the theme's name is represented by a phrase or an adjective, which gives the analyst an indication to attach it to the theme. In the worst case the requirement and the theme could be related by implied actions: actions that are activated as a consequence of other actions [5]. The same happens when crosscutting relationships are identified. According to Theme/Doc, shared requirements are potential indicators of crosscutting. Having a shared requirement means that two concerns are present in the text [10]. This suffers from the same drawback of requiring unambiguity.

Baniassad and Clarke [3] have shown how Theme/Doc analysis of actions helps to solve some ambiguities and how a synonym dictionary helps in the case of multiple terms referring to the same concept. In our experience, the problem goes deeper than the use of synonyms: we not only have some words that are written in different way, sometimes ideas are equivalent but explained differently.

Considering the kind of requirements we face, we consider two options to resolve ambiguities. The first one is to rewrite all the requirements, normalising them to use the same vocabulary; the second one is to use domain knowledge to associate requirements to the corresponding themes directly. Due to large number of requirements and presence of multiple sources the first option is not feasible, we therefore opt for the second. Grouping requirements into concerns based on domain knowledge is not new [4, 15]. Our experience is that the resulting concerns are useful as they can be easily discussed with domain experts.

As a consequence of doing a domain knowledge based analysis of our requirements and concerns we also noticed that the information contained in the original requirements, *in some cases*, needs to be combined with domain knowledge. This in order to generate new requirements that are more suitable for understanding concern relationships. This is similar to the approach proposed by Bar-On et al. [5], where implied actions are used to generate new *derived requirements*.

4.4 Possible Extensions to Theme/Doc

Given our experience using Theme/Doc in the SM domain, we propose possible enhancements that should address the limitations we encountered:

Granularity.

In order to improve the information of requirements participating in a crosscutting relationship, it should be possible to add the IDs of the affected requirements (or some kind of quantifier) in the form of tags attached to the crosscutting arrows. It would thus be possible to clearly express which are the affected requirements.

Expressing Interactions.

Theme/Doc's graphical notation needs to be extended to support expressing interactions. One way to do this is to include relationships between concerns, as we did in Fig. 1.

Adaptability to domain-specific requirements.
To fix this we need to consider a time for adding domain specific knowledge, this may imply adding some specific task in the process of Theme, probably during the requirements processing (where requirements are split, removed or added).

5. EVALUATION OF MDSOCRE

MDSOCRE (Multidimensional Separation of Concerns in Requirements Engineering) is the evolution of a line of AORE approaches such as PreView and ARCaDe [18]. As it appears to provide the most expressive and flexible constructs for binding crosscutting concerns to base concerns, we choose it as the second case in our study.

5.1 Brief overview of MDSOCRE

Multidimensional Separation of Concerns in Requirement Engineering [15], is a refinement of the ARCaDe approach [18]. In contrast to Theme, it does not provide visualisation facilities. MDSOCRE treats the concerns in a uniform fashion, regardless of the nature of the requirement (functional or non-functional). It makes it possible for the requirement engineer to choose a subset of requirements to observe the influences on each other and to analyse crosscutting behaviour.

Conflicts referring to contradictory concerns are detected and handled using contribution matrices. In such a matrix rows and columns identify concerns and the cells denote how the concerns contribute to the other (negative contributions denote conflicts). These matrices help in the decision process of which (parts of) features will be implemented. Conflicts in MDSOCRE differ slightly from our definition in Sect. 3.2 (taken from [20]). In our case, concerns are not a subject of negotiation, as all are required by some standard or regulation. We must however check that at runtime conflicting concerns are not simultaneously active.

MDSOCRE also provides support for *meta concerns*: generic concerns that are instantiated for specific systems. The most important feature of meta concerns for us is their capability for expressing commonly related concerns. We will use this to express interactions in Sect. 5.3.

At an implementation level, MDSOCRE uses XML to express requirements and composition rules, we show an example next.

5.2 Successful uses of MDSOCRE

Figure 4 shows how some of the concrete concerns of our domain are expressed in this approach. The `Concern` tag is composed of several requirements which are surrounded by the `Requirement` tag. A requirement can be referenced by its identifier (*id*) and can contain nested subrequirements. Due to space constraints we do not include the detailed listing of all concerns here[3].

Composition rules are used to express crosscutting relationships. Figure 5 shows composition rules, consisting of a `Constraint` tag that defines how the base requirements are constrained by aspectual requirements. The `Constraint` tag

[3]The complete set can be found online at `http://pleiad.cl/research/adi`

```
<Concern name="Game">
  <Requirement id="1">Slot machines have 5 reels which spin when a
      button (PLAY) is pushed </Requirement>
  <Requirement id="2">Reels spin when play button is pressed
      </Requirement>
  <Requirement id="3">Prizes are awarded according to pay table.
      </Requirement>
  <Requirement id="4">A slot machine has one or mode devices for
      entering money. </Requirement>
  <Requirement id="5">As money is inserted credits are "assigned" to
      the player.</Requirement>
  <Requirement id="6"> A slot machine must provide some means for
      cashing the credits out. It could be a ticket printer or a coin
      hopper. </Requirement>
</Concern>
<Concern name="Game Recall">
  <Requirement id="1">Information on at least the last ten (10) games is
      to be always retrievable on the operation of a suitable external
      key-switch, or another secure method that is not available to the
      player. </Requirement>
  <Requirement id="2">Last play information shall provide all
      nformation required to fully reconstruct the last ten (10) plays.
      All values shall be displayed; including the initial credits, credits
      bet, and credits won, payline symbol combinations and credits
      paid whether the outcome resulted in a win or loss [...]. This
      information should include the final game outcome, including all
      player choices and bonus features.</Requirement>
</Concern>
```

Figure 4: Game and Game Recall concerns

has *actions*, *operators* and *outcome* elements, used to express in detail how the base is affected. The action and operator tags informally describe how the base concern is constrained, imposing conditions in the composition. The operators express temporal intervals, temporal points or restrictions between sets of concerns. The outcome tags (*satisfied* and *fulfilled*) define the result of such composition, to assert that constraints have been successfully imposed. For detailed information about the semantics of these elements, refer to [15].

The first composition rule of Fig. 5 shows how the *Meters* concern crosscuts the *Game* concern. In this example we have used the outcome action "fulfilled", because there is no other set of requirements to be satisfied. Otherwise the action value should have been "satisfied" and the set of requirements that are satisfied. This would be the case of the *Error Condition* composition, since when such a condition is detected an action must be taken, i.e. a requirement has to be satisfied after the constraints have been applied. A concrete example of this is the *door open* error condition: it has to be reported after it has been detected and the SM should lock up until the condition is resolved.

The granularity of the approach is adequate for our case study, since it is possible to clearly state which requirements are affected. The flexibility provided by the parametrised `constraint` tag helps to express different variants of crosscutting relationships. For example, we combine actions and operators to document the interactions. We used the action *ensure* and the operator *with* to represent a *Dependency* interaction. This follows the informal definition by Moreira et. al. [15], that says that a certain condition for a requirement that is needed actually exists. We used the action *provide* and the operator *for* for *Reinforcement*, as it specifies additional features to a set of concern requirements.

5.3 Limitations of MDSOCRE

```
<Composition>
  <Requirement concern="Meters" id="4">
      <Constraint action="enforce" operator="on">
          <Requirement concern="Game" id="3" />
      </Constraint>
      <Outcome action="fulfilled"/>
  </Requirement>
  <Requirement concern="Meters" id="3">
      <Constraint action="ensure" operator="with">
          <Requirement concern="G2S" id="2"/>
      </Constraint>
      <Outcome action="fulfilled"/>
  </Requirement>
</Composition>
<Composition>
  <Requirement concern="Error Condition" id="3">
      <Constraint action="provide" operator="for">
          <Requirement concern="G2S" id="4" />
      </Constraint>
      <Outcome action="fulfilled"/>
  </Requirement>
  <Requirement concern="Error Condition" id="3">
      <Constraint action="enforce" operator="on">
          <Requirement concern="Game" id="2" />
      </Constraint>
      <Outcome action="satisfied">
          <Requirement concern="Error Condition" id="2" />
      </Outcome>
  </Requirement>
</Composition>
```

Figure 5: Composition rules for Meters and Error Condition concerns

The actions and operators included in the composition rules only describe relationships between the crosscutting concern and the selected base concern. As we explained in Sect. 3.2, interactions occur even between concerns without a crosscutting relationship. In our case we need to express somehow that *G2S* depends on the existence of *Meters* to report this information and also that having *Error Conditions* could reinforce the functionality of *G2S* enabling it to report a new set of events (errors). These interactions as well as mutex (see Sect. 3.2) are not explicitly supported by this approach.

As a workaround we have combined pairs of actions and operators, for example: the action *ensure* and the operator *with* to represent a *Dependency* in the case of *Meters* and *G2S*, and the action *provide* with the operator *for* to represent *reinforcement* of *Error Conditions* and *G2S*.

This solution however has two downsides:

1. It forces the use of compostion rules even when no crosscutting is present, which seems contradictory with the original purpose of composition rules expressed by the authors: "they describe how a concern cuts across other concerns..." [15]

2. The expressiveness of our combinations is not optimal, as it is not easy to map the different interaction types with pairs of actions and operator. Consider for instance *provide for* compared to the word *"reinforce"*. *Reinforce* makes it explicit that the interaction is a positive influence to the other aspect, but we have to use *provide for* which is only a way to try to represent this idea.

An alternative would be the use of meta concerns, which seem to be a natural place to store information regarding interactions. Meta concerns are not exactly aimed at this purpose, but with little extension they can support the different kinds of interactions. The drawback here is a conceptual mismatch: meta concerns were designed to document generic concerns, but in our case interactions are manifest in concrete concerns. We therefore did not use this alternative.

5.4 Possible Extensions to MDSOCRE

Unfortunately, MDSOCRE does not natively support the notion of interactions. Although they can be expressed using a combination of actions and operators, it is not a clean solution, as interactions are not explicitly documented. Alternatively, the meta concern facility can be used to store information regarding interactions, but it is concrete information that would be stored in an artifact aimed at expressing generic information regarding concerns.

We believe that a new relationship between concerns, aimed at documenting interactions, is needed. The new relationship would enable us to express interactions between concrete concerns, as well as between meta concerns.

6. RELATED WORK

We have encountered two other comparative studies for AORE approaches but these however do not consider aspectual interactions. Sampaio et al. [19] analyse the speed of the requirement analysis process and the quality of the output. Their real world example (19 pages of requirements specification), is considerably smaller than ours, (about 150 pages [11, 12, 16]). Chitchyan et. al. [9] use several comparison criteria (identification of concerns, composability, decision support, traceability, evolvability and scalability). This work is more conceptual, as it compares the approaches without applying them to a concrete example, instead it considers the mechanisms provided by each approach in light of the criteria mentioned before.

AORE approaches that consider *conflict* resolution as part of their methodology [7, 18] help stakeholders decide on which concern to implement. In our case however *all* conflicting concerns must be implemented. Conflicts need to be documented so that interactions are considered at design time.

In [23] Whittle et al. present an approach called MATA, based on model transformation where weaving is viewed as a special case of graph transformation. MATA provides support for conflict and dependency detection, based on critical pair analysis. The objective of this detection phase is to order composition. More recent work, by Chitchyan et al. [8] (not available at the time when we started modelling our requirements) moves the focus towards a semantic analysis of requirements. Here requirements are annotated and then composition rules can be expressed using semantic queries. This approach enables to automatically detect certain conflicts, with the aim of removing them. As we stated before, we need to document the interactions, not removing (all of) them.

In this work our focus lies on interactions between aspects, and capturing them at requirement level. Multiple approaches for capturing requirements using aspects exist that however do not provide support for interactions [3, 14, 22] we therefore did not include them in our case study. Some other approaches, such as AORA [6] provide documentation of dependencies, but nothing is said regarding mutex or reinforcement interactions.

7. CONCLUSIONS AND FUTURE WORK

This paper presents our applicability study of two AORE approaches: Theme/Doc [2] and MDSOCRE [15], in the Slot Machine Domain. We focused mainly on the expressiveness of these approaches in terms of interactions between

requirements. We found that both approaches lacked complete support for our case and proposed some extensions to both approaches that might address this.

From our analysis we conclude that, compared to Theme/Doc, MDSOCRE has a strong point in that it allows to specify the composition of concerns in detail. Both approaches do not provide explicit support for expressing interactions. We had to simulate this by defining new relationships between aspectual concerns, which was only possible in MDSOCRE. Using parametrised composition rules allowed us to express interactions in an indirect way.

Finally, we noticed a considerable difference in the process for attaching requirements to their concerns. Theme/Doc relies on the analysis of the text of requirements, searching for the concern name, while MDSOCRE relies on analyst's domain knowledge. As we have different sources with different terminology, we found the MDSOCRE approach more suitable for our needs.

Our future work consists of applying the enhancements we propose in order to fully evaluate them, and to proceed with the design phase of the application.

8. REFERENCES

[1] Recommended practice for software requirements specifications. *IEEE Std 830-1998*, 1998.

[2] E. Baniassad and S. Clarke. Finding aspects in requirements with theme/doc. In *Early Aspects Workshop at AOSD*, March 2004.

[3] E. Baniassad and S. Clarke. Theme: An approach for aspect-oriented analysis and design. In *ICSE '04: Proceedings of the 26th International Conference on Software Engineering*, pages 158–167, Washington, DC, USA, 2004. IEEE Computer Society.

[4] E. Baniassad, P. C. Clements, J. Araujo, A. Moreira, A. Rashid, and B. Tekinerdogan. Discovering early aspects. *IEEE Softw.*, 23(1):61–70, 2006.

[5] D. Bar-On and S. Tyszberowicz. Derived requirements generation: The dras methodology. *Software Science, Technology and Engineering, IEEE International Conference on*, 0:116–126, 2007.

[6] I. Brito and A. Moreira. Integrating the nfr framework in a re model. In *Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design, workshop of the 3rd International Conference on Aspect-Oriented Software Development*, 2004.

[7] I. S. Brito, F. Vieira, A. Moreira, and R. A. Ribeiro. Handling conflicts in aspectual requirements compositions. *Transactions in Aspect-Oriented Software Development*, 3:144–166, 2007.

[8] R. Chitchyan, A. Rashid, P. Rayson, and R. Waters. Semantics-based composition for aspect-oriented requirements engineering. In *AOSD '07: Proceedings of the 6th international conference on Aspect-oriented software development*, pages 36–48, New York, NY, USA, 2007. ACM.

[9] R. Chitchyan, A. Rashid, and P. Sawyer. Comparing requirement engineering approaches for handling crosscutting concerns. In J. Araújo, A. D. Toro, and J. F. e Cunha, editors, *WER*, pages 1–12, 2005.

[10] S. Clarke and E. Baniassad. *Aspect-Oriented Analysis and Design. The Theme Approach.* Object Technology Series. Addison-Wesley, Boston, USA, 2005.

[11] Gaming Laboratories International. *Gaming Devices in Casinos*, 2007. Available at: http://www.gaminglabs.com/.

[12] Gaming Standard Association. *Game to Server (G2S) Protocol Specification*, 2008. Available at: http://www.gamingstandards.com/.

[13] J. C. Grundy. Aspect-oriented requirements engineering for component-based software systems. In *RE '99: Proceedings of the 4th IEEE International Symposium on Requirements Engineering*, pages 84–91, Washington, DC, USA, 1999. IEEE Computer Society.

[14] I. Jacobson and P.-W. Ng. *Aspect-Oriented Software Development with Use Cases (Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, 2004.

[15] A. Moreira, A. Rashid, and J. Araujo. Multi-dimensional separation of concerns in requirements engineering. In *Proc. 13th IEEE International Conference on Requirements Engineering*, pages 285–296, 29 Aug.–2 Sept. 2005.

[16] Nevada Gaming Commission. *Technical StandardsăForăGamingăDevices And On-Line Slot Systems*, 2008. Available at: http://gaming.nv.gov/stats_regs.htm.

[17] A. Rashid and A. Moreira. Domain models are NOT aspect free. In *ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems (MODELS06)*, volume 4199 of *Lecture Notes in Computer Science*, pages 155–169. Springer Verlag, October 2006.

[18] A. Rashid, A. Moreira, and J. Araújo. Modularisation and composition of aspectual requirements. In *AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 11–20, New York, NY, USA, 2003. ACM.

[19] A. Sampaio, P. Greenwood, A. F. Garcia, and A. Rashid. A comparative study of aspect-oriented requirements engineering approaches. In *ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, pages 166–175, Washington, DC, USA, 2007. IEEE Computer Society.

[20] F. Sanen, E. Truyen, B. D. Win, W. Joosen, N. Loughran, G. Coulson, A. Rashid, A. Nedos, A. Jackson, and S. Clarke. Study on interaction issues. Technical Report AOSD-Europe Deliverable D44, AOSD-Europe-KUL-7, Katholieke Universiteit Leuven, 28 February 2006 2006.

[21] P. Tarr, H. Ossher, W. Harrison, and J. Stanley M. Sutton. N degrees of separation: multi-dimensional separation of concerns. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 107–119, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.

[22] J. Whittle and J. Araújo. Scenario modelling with aspects. *IEE Proceedings - Software*, 151(4):157–172, 2004.

[23] J. Whittle and P. Jayaraman. Mata: A tool for aspect-oriented modeling based on graph transformation. pages 16–27, 2008.