Experience Report: AORE in Slot Machines

Arturo Zambrano, Johan Fabry and Silvia Gordillo

Abstract In the context of an industrial project in the domain of slot machines we needed to perform Aspect Oriented Requirements Engineering, with a special emphasis on dependencies and interactions among concerns. The critical importance of interactions in this domain demanded explicit and detailed documentation of all interactions. We evaluated two AORE approaches: Theme/Doc and MDSOCRE, to establish their applicability in our setting. In this work we report on our experience, showing successful uses of both approaches and also where they fall short. To address these limitations, we have proposed some enhancements for both approaches and we present them here as well.

1 Introduction

To have Aspect-Oriented Requirements Engineering (AORE) gain the acceptance of the software development industry and become a mainstream practice for requirement engineering, it is necessary to demonstrate its power against industrial problems. This work is intended to be a contribution in that direction.

In the context of an industrial project we are re-implementing the software that runs on the casino gambling device best known as a slot machine (SM). Due to previous experience with this software we know that there are an important amount of

Arturo Zambrano
LIFIA, Facultad de Informática, Universidad Nacional de Plata, 50 y 115 La Plata, Argentina. e-mail: arturo@lifia.info.unlp.edu.ar

Johan Fabry
Pleiad Lab, DCC, Universidad de Chile, Blanco Encalada 2120, Santiago, Chile. Partially funded by FONDECYT project 1090083. e-mail: jfabry@dcc.uchile.cl

Silvia Gordillo
LIFIA, Facultad de Informática, Universidad Nacional de Plata, 50 y 115 La Plata, Argentina. e-mail: gordillo@lifia.info.unlp.edu.ar

crosscutting concerns in slot machine applications. Moreover, many of these concerns interact with each other. We therefore have opted to use Aspect-Oriented Software Development in our new implementation.

Being aware of the critical importance of interactions in this domain, we have focused early in the development cycle, i.e. in the AORE phase, on interaction modeling. Our specific objective for this step is to document all interactions explicitly. This information would then be used later in the design and implementation phases. We therefore require the result of the modelling process to be a consistent model of the requirements, containing detailed and explicit interactions. To accomplish this, we needed to be able to rely on expressive mechanisms in the selected modeling techniques for this phase. To establish their suitability to our needs we therefore performed an evaluation of two existing AORE approaches.

We elected to evaluate performing requirements engineering using both the Theme/Doc approach [3] and Multidimensional Separation of Concerns for Requirements Engineering (MDSOCRE) [8], 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 their publication record.

In this text we report on our experiences evaluating the above two approaches, and include proposals for extending them where they fall short. Our evaluation has been reported in more detail in [13]. In this chapter, we focus on the more salient points of the evaluation and add our proposals for extension. Briefly put, our evaluation has shown that both of the approaches we evaluated enable us to express the requirements, but neither of them satisfies our needs with regard to the specifications of interactions. We were however able to extend both approaches such that these limitations were overcome.

This chapter is organised as follows: Sect. 2 characterizes the slot machine domain, its concerns, and requirements. In Sect. 3 we present the results of applying the two AORE approaches to the SM domain, including report of their shortcomings and proposals for extensions. Sect. 4 presents the conclusions for our work.

2 Requirements for Slots Machines

A slot machine (SM) is a gambling device. It has five reels which spin when a *play* button is pressed. A 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 a *cashout*) as coins, tickets or electronic transfers.

Requirements for the SM domain are defined in different documents: Regulations (for each jurisdiction), standards (documents released by certification laboratories) and protocol specifications (technical documents for interoperability). These are written by different stakeholders, with diverse interests and backgrounds. This results in a large set of documents using multiple terms for describing the same ob-

ject, action or event. 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 the Nevada regulations [9] and the GLI standard [7]; some of the conditions specified by the regulations match, but others are defined by just one of them.

A notable characteristic of communication protocol requirements, which has an impact on requirements modeling, is that they are divided in topics and that part of the communication functionality is optional. As we will see later, these optional requirements are the source of one of the aspectual interactions we need to deal with.

2.1 Crosscutting Concerns in the Slot Machine Domain

Based on our experience in the domain, we organised the requirements as follows, where Game is a base concern and the rest are crosscutting concerns[1]. As the domain is complex, there may be other possible concern decompositions. We choose this one because, according to our experience and observations, it properly modularises the different required features of slot machines and shows the interactions that are at the core of this evaluation.

Game: This is the basic logic of a gambling device at a casino. 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 (hundreds 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 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). This concern defines the kinds of data, the format and when data must be exchanged between the SM and RS. Several communication protocols with similar functionality exist. In this work we will refer to the most widely used protocols in the SM industry: *Game to Server protocol* (G2S) and *Proprietary Communication Protocol* (PCP[2]).

---

[1] We use the terms *Base* and *crosscutting concerns* as usual in the AOSD community.

[2] Licensing issues prevent us to use the real protocol name and disclosing implementation details.

Demo The demo concern contains the requirements specifying how the game behaves in this mode. Playing the game in demo allows testing hardware and software works, simulating events such as entering money or winning a prize.

2.2 Interactions in the Slot Machine Domain

We based our investigation, and therefore this discussion on the AOSD-Europe technical report that gives an overview on aspect interactions [11]. In this report, the authors classify aspect interactions into dependency, conflict, mutex (mutual exclusion) and reinforcement. Dependency is the case where one aspect explicitly needs another to work correctly. A conflict between two aspects happens when they work correctly in isolation, but the presence of both at the same time negatively influences the behavior of the system. A mutual exclusion (mutex for short) occurs when two aspects implement the same functionality, but only one of them can be used at a time. Reinforcement is a positive interaction where an aspect influences the correct working of another, allowing it to provide extended functionality. Note that, even though the consequence of mutex and conflict are the same (just one of the conflicting aspects can be active at a time) there is a semantic difference: mutex applies to aspects that implement similar behavior, while conflict is more general and applies to any kind of incompatibility between aspects. We found this classification to match the kinds of interactions we observed in the SM domain.

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* concerns (Proprietary Protocol and G2S Protocol). 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 concern. 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.

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. The notation in this figure is an ad-hoc mechanism to analyse the relationships between concerns that we will try to model in the following sec-

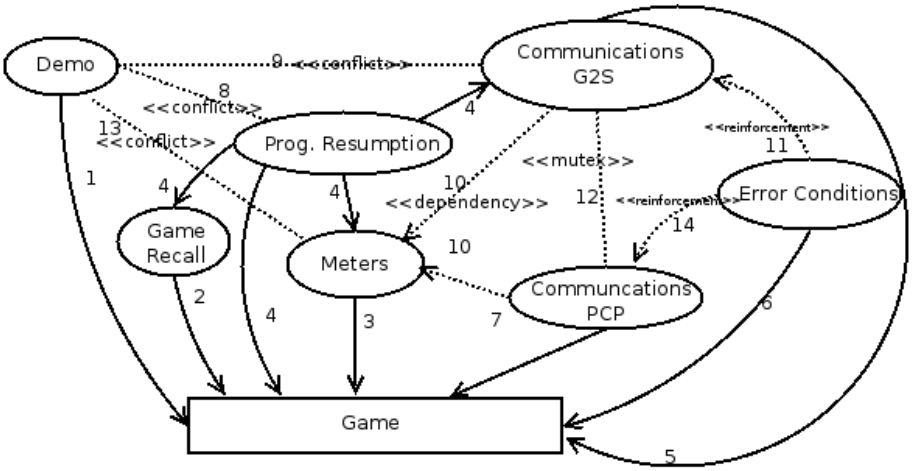

Fig. 1: Concern interactions. Regular arrows indicate crosscutting, dashed arrows indicate interactions between concerns, tagged with UML-like stereotypes.

tions using well known AORE approaches. The base concern (Game) is depicted by a box and crosscutting concerns by ovals. Relationships 1 to 7 are crosscutting (solid arrows). For each, there is a crosscutting concern where one or more requirements cut across several requirements on the base requirement (where the arrow ends). For example, consider the relationship between *Error Conditions* and *Game*, where the behaviour associated with error conditions needs to be woven into the game behaviour. Requirements in *Game* that could raise an error condition vary: a bill inserted, the printer is out of paper, tilt, a door opened, etc.

In Fig. 1, relationships 8 to 14 are interactions, which are depicted by dotted lines. Dependency and reinforcement are asymmetric, so the arrowhead indicates the direction of the relationship, while mutex and conflict are symmetric, so no arrowhead is used. Table 1 describes them and presents potential consequences of not considering interactions during the design/implementation phases. We use an informal notation here and will later evaluate how different AORE approaches perform while trying to model such information more formally.

2.3 Selected Requirements

In this text, due to space limitations, we only use three concerns to illustrate our work. We focus on the crosscutting relationship between Meters and Game, and the *dependency* of G2S on Meters. Table 2 presents an extract of the requirements for these concerns. We refer to [13] for a more complete treatment.

Table 1: Interaction consequences.

Interaction	Description	Consequence if not considered
8. Conflict between *Demo* and *Program Resumption*	The demo mode fires *fake* events that must not be counted nor restored after program interruption.	Wrong data is loaded after a reboot while in Demo, accounting mismatches, auditing errors.
9. Conflict between *Demo* and *G2S*	Both concerns cannot be active, because demo fires fake events that must not be reported.	Inconsistent accounting reports including *fake* data.
10. Dependency of *G2S* and *Prop. Protocol* on *Meters*	Data reported to the RS is stored or can be derived from meters. Communication protocols need the meters to be up to date in order to accomplish its purpose.	Communication protocols could report old data if meters are not working.
11. Reinforcement of *G2S* with *Error Conditions*	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.	Real time events are not reported when available. Casino operator cannot efficiently react to situations such as: coin-tilt, handpay, stacker full.
12. Mutex between *G2S* and *Proprietary Protocol*	There is overlapping functionality defined in the requirements of both protocols. Therefore they cannot be active at the same time.	Overlapping features of both can interfere. For example using both to keep the time in sync between the SM and the RS may render the time of the SM inconsistent.
13. Conflict between *Demo* and *Meters*	Data generated during demo mode must not affect meter values.	Demo plays may result in inconsistent accounting information if they are counted by the meters.
14. Reinforcement of *Prop. Protocol* with *Error Conditions*	Similar to 11.	Similar to 11.

3 Application of AORE

In order to deal with requirements and concerns in the SM domain we applied two well known AORE approaches: Theme/Doc and MDSOCRE. This resulted in the identification of some limitations and proposed extensions. We discuss this here, first focusing on Theme/Doc and then on MDSOCRE.

3.1 Application of Theme/Doc

Theme/Doc [2] is the requirement analysis part of the Theme approach [3, 6]. 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, as in our setting it is error prone due to ambiguities (see Sect. 3.1.2). Instead we use the concerns we identified in Sect. 2.1 as themes.

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: 1) the requirement cannot be split in order

Table 2: Requirements for Game, Meters and G2S concerns.

Game			
GM-1	Slot machines have 5 reels.	GM-4	A slot machine has one or more devices for entering money.
GM-2	Reels spin when play button is pressed.	GM-5	As money is inserted credits are "assigned" to the player.
GM-3	Prizes are awarded according to a pay table.	GM-6	A slot machine must provide some means for cashing the credits out. It could be a ticket printer, a coin hopper.
Meters			
M-1	Credit meter: shall at all times indicate all credits or cash available for the player to wager or cashout.	M-3	Accounting Meters: *Coin In*: [...] a meter that accumulates the total value of all wagers [...]. *Games-played*: accumulates the number of games played; since power reset, since door close and since game initialisation.
M-2	Credit Meter Incrementing: The value of every prize (at the end of a game) shall be added to the player's credit meter [....]. The credit meter shall also increment with the value of all valid coins, tokens [..].	M-4	Meters should be updated upon occurrence of any event that must be counted, including: play, cashout, bill in, coin in.
M-5	G2S meters are: *gamesSinceInitCn* Number of games since initialisation. *WonCnt*: Number of primary games won by the player. *LostCnt*: Number of primary games lost by the player.		
Communication: G2S			
G2S-1	The G2S protocol is designed to communicate information between an SM, and one or more host systems.	G2S-4	The device can generate an event in a unsolicited manner or in response to a host command
G2S-2	Meter information can be queries by a host in real-time or a host may set a periodic subscription to cause the SM to send selected meters[..]	G2S-5	Current time-stamp can be set by the host.
G2S-3	Information provided by the SM is used for audit purposes.	G2S-6	Command GetGameRecallLog is used by a host to request the contents of a transaction log of last plays from a SM.

to avoid tangling, 2) one of the themes dominates the requirement, 3) the dominant theme is triggered by events in the base theme, and 4) the triggered theme is externally fired in multiple situations [6]. Note that for Theme/Doc, the term *dominant* refers to the potentially crosscutting concern, *i.e.,* which contains the requirement that cuts across other requirements.

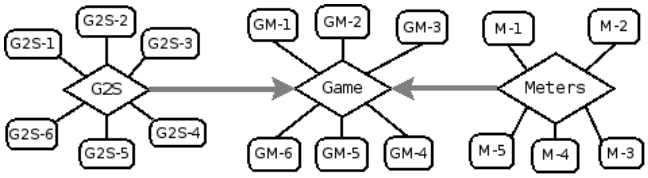

Fig. 2: *Game*, *Meters*, and *G2S* concerns expressed using the Theme/Doc notation.

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 grey 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*, and *G2S* concerns are represented along with their requirements and crosscutting relationships.

3.1.1 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 reasonably documented, enabling us to easily identify which concern is playing the base and/or the aspectual role. Furthermore, it is possible to express all the crosscuting relationships shown in Fig. 1, although we cannot include them here due to space limitations.

3.1.2 Limitations of Theme/Doc

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

Granularity: As explained before, grey 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. 2.2 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 [12]. 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. 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: In our case study there is no single requirements specification unifying all the sources and we are faced with significant ambiguity. This in contrast to an ideal requirements specification that is complete, unambiguous, verifiable, consistent, modifiable and traceable [1]. This 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 based on requirements from a real-world problem and it is worthwhile to examine the impact of this.

The ambiguity we face affects the mechanism proposed in Theme/Doc to assign requirements to themes, and to identify potential crosscutting themes. In the most ambiguous case, the requirement and the theme could be related by implied actions: actions that are activated as a consequence of other actions [5]. Unfortunately, the ambiguities we found cannot be solved by using a synonym dictionary as proposed in [3].

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. Due to the large number of requirements and presence of multiple sources only the second option is feasible, and moreover is a well known practice [4, 8].

As a consequence of doing a domain knowledge based analysis of our requirements, new requirements that are more suitable for understanding concern relationships, may appear. This is similar to the approach proposed by Bar-On et al. [5], where implied actions are used to generate new *derived requirements*.

3.1.3 Extensions to Theme/Doc

Quantification Labels for Granularity In order to tackle the granularity problem, we introduce the concept of *quantification labels*, which are tags that allows us to clearly specify which requirements participate in a crosscutting relationship (and are also permitted in the interaction relationships we introduce below). A quantification label is an expression referring to a *base concern* and a *crosscutting concern*. Fig. 3 shows an example where the *Meters* concern crosscuts the *Game* concern, we can see here that requirement M-4 crosscuts requirements GM-2 to GM-6.

Quantification labels allow us to specify which requirements are involved in a given crosscutting (and interaction) relationship, from both sides: the crosscutting concern and the base concern. It has two parts separated by a colon:

Crosscutting requirements IDs: this is a list, a range, or the keyword *all* that indicates which requirements are crosscutting in the concern where the arrow has its origin.

Base concern requirements IDs: this is a list, a range, or the keyword *all* that indicates which requirements are the requirements affected by the crosscut concern (the destination of the arrow).

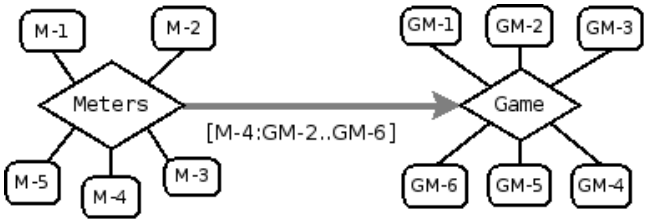

Fig. 3: Quantification labels applied to the crosscutting relationship between Meters and Game concerns.

Interaction Relationships In order to properly express the interactions between concerns, we added a new kind of relationship to Theme/Doc. The new interaction relationship is denoted using a dashed arrow. The arrow also has a label indicating the kind of interaction. Quantification labels can be used along with interaction relationships, this allows to clearly state the requirements interacting for each concern. The interaction relationship can be symmetrical (mutex or conflict) or directional (dependency and reinforcement).

Fig. 4 shows a dependency between the *G2S* concern, which needs the information stored by the *Meters* concern. The dotted line indicates the interaction, which in this case is directional. Quantification labels are also included to indicate the requirements participating in the interaction. The "In Balance Meters" label is a derived requirement, which we explain below.

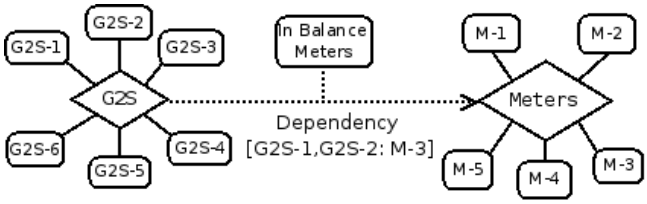

Fig. 4: Dependency of G2S concern on Meters concern.

Ambiguity of Requirements In our setting, requirements disambiguation needs to be performed by domain experts. The Theme/Doc methodology establishes specific steps for requirements processing (e.g. split, add and remove) [3]. We propose to add a dedicated step that performs disambiguation before performing the existing processing steps. As a result, during this step new (derived) requirements may arise as shown in Fig. 4. In this case the derived requirement states that meters need to be updated in consistent sets, so that they are reported to the accounting reporting system when they are in balance. This derived requirement is represented by the *In Balance Meters* label in Fig. 4.

3.2 Application of MDSOCRE

MDSOCRE (Multidimensional Separation of Concerns in Requirements Engineering) is the evolution of a line of AORE approaches such as PreView and ARCaDe [10]. 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 behavior.

In contrast to Theme, MDSOCRE does not provide visualisation facilities. Conflicts referring to contradictory concerns are detected and handled using contribution matrices. Conflicts in MDSOCRE differ slightly from our definition in Sect. 2.2 (taken from [11]). 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 uses XML to express requirements and composition rules. For example, listing 1 shows how the Game and Meters concerns are expressed in this approach. The `Concern` tag is composed of several requirements which are indicated by the `Requirement` tag. A requirement can be referenced by its identifier (*id*) and can contain nested sub-requirements. Furthermore, concerns and requirements can be related through composition rules, using the `Composition` element, which we will explain in the following section.

Listing 1: Concerns written using MDSOCRE

```
<Concern name="Game">
  <Requirement id="1"> A slot machines has 5 reels. </Requirement>
  <Requirement id="2"> Reels spin when the play button is pressed.</
      Requirement>
  <Requirement id="3"> Prizes are awarded according to a pay table.
  </Requirement>
  <Requirement id="4"> A slot machine has one or more 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 means for cashing the
      credits out.</Requirement>
</Concern>
<Concern name="Meters">
   <Requirement id="1"> Credit meter: shall at all times indicate all credits
or cash available for the player to wager or cashout
 </Requirement>
  <Requirement id="2"> Credit Meter Incrementing: The value of every
prize [...futher details omitted ... ]
  </Requirement>
  <Requirement id="3"> Accounting Meters: Coin In: a meter that
accumulates the total value of all wagers [... omitted ...]. </Requirement>
   <Requirement id="4"> Meters should be updated upon occurrence of any event
       that must be counted, including: play, cashout, bill in, coin in.
 </Requirement>
</Concern>
```

3.2.1 Successful uses of MDSOCRE

Listing 2: Compositions written using MDSOCRE

```
<Composition>
 <Requirement concern="Meters" id="4">
   <Constraint action="enforce" operator="on">
        <Requirement concern="Game" id="3,4,5,6" />
   </Constraint>
   <Outcome action="fulfilled"/>
 </Requirement>
</Composition>
```

Composition rules are used to express crosscutting relationships. Listing 2 shows a composition rule, consisting of a `Constraint` tag that defines how the base requirements are constrained by aspectual requirements. The `Constraint` tag 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. We shall use these in Sect. 3.2.2 and 3.2.3, for more detailed information about them we refer to [8].

The composition rule in Listing. 2, 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.

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 use the action *ensure* and the operator *with* to represent a *Dependency* interaction. This follows the informal definition by Moreira et. al. [8], that says that a certain condition for a requirement that is needed actually exists. We use the action *provide* and the operator *for* for *Reinforcement*, as it specifies additional features to a set of concern requirements.

3.2.2 Limitations of MDSOCRE

No Support for Interactions 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. 2.2, 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. 2.2) are not explicitly supported by this approach. Note that conflict is not supported in any way, as they are supposed to be removed through negotiation. 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 composition 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...” [8]
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 interact on is a positive influence to the other aspect, but we have to us *provide for* to represent this idea.

No Support for Unification As mentioned before in Sect. 3.1.2, in our setting there are multiple and ambiguous requirement documents and, as in Theme/Doc, this raises unification issues. Some concerns, such as *Meters*, are defined in many requirements in several of these documents. This makes it difficult to trace the complete definition of meters (which is necessary for the design and implementation). Rewriting all the requirements referring to meters, to condense them into one piece of requirements is not feasible due to the large number of requirements. Besides this, it is difficult to maintain the merged version of the requirements once one of the sources evolves. We conclude that for MDSOCRE we need a way of keeping related requirements linked without coupling them, so that they can evolve at their own pace.

3.2.3 Extensions of MDSOCRE

Explicit interaction compositions In order to provide explicit support for interactions, we extended MDSOCRE. After evaluating several possible extensions that are not included here due to space limitations, we decided to extend the `Composition` element with a new `Interaction` element. It can be parametrised with the specific interaction type. The `Interaction` element is contained in a `Requirement` element, and itself includes at least another `Requirement` element, as can be seen in Listing 3. The interaction direction goes from the outer element to the inner one. For example, Listing 3 is read as follows: *requirements 1 and 2 from the G2S communication protocol depends on requirement 3 from the Meters concern.* Note that this order only applies to the directional interactions *dependency* and *reinforcement*.

Listing 3: Explicit interaction support for MDSOCRE

```
<Composition>
  <Requirement concern="G2S" id="1,2">
    <Interaction type="dependency">
      <Requirement concern="Meters" id="3"/>
    </Interaction>
  </Requirement>
</Composition>
```

Cross-references for Linked Requirements The different requirement sources complement each other. Hence there is no unique and complete piece of text that allows us to produce a full design for certain requierements. Therefore, to enable an

unabridged description of these requirements, we extended MSDOCRE with cross-references. We added a new attribute, called `seeAlso`, to the `Requirement` element. The value associated with this attribute is a list of requirements IDs where additional information is present. The `seeAlso` allows to relate all the requirements defining one concept (see Listing 4). These references are intended to be used in a single concern. As we mentioned before, this issue also manifests itself in Theme, but have not been able to solve it without overly cluttering the diagrams.

Listing 4: Cross references extension

```
<Concern name="Meters">
 <Requirement id="1" seeAlso="3,4,5"> Credit meter: shall at all times
      indicate all credits or cash available for the player to wager or
      cashout. </Requirement>
 <Requirement id="2"> Credit Meter Incrementing: The value of every prize
      [...] </Requirement>
 <Requirement id="3" seeAlso="1,4,5"> Accounting Meters: Coin In: a meter that
       accumulates the total value of all wagers . Games-played: [...]
 </Requirement>
 <Requirement id="4" seeAlso="1,3,5"> Meters should be updated upon occurrence
       of any event that must be counted, including: play, [...].
</Requirement>
<!-- From G2S Docs-->
 <Requirement id="5" seeAlso="1,3,4"> Some G2S meters are: gamesSinceInitCn
      Number of games since initialisation. [...]
 </Requirement>
</Concern>
```

4 Conclusions

In our work, we evaluated two well-known AORE approaches in an industrial setting – the slot machines (SM) domain – where many functional crosscutting concerns are present. This domain is furthermore characterized by aspectual interactions, and the legal applicability of several large requirement documents that have ambiguity issues. In our previous experience, developing this software and not considering aspectual interactions led to costly bugs in production.

When re-implementing the SM software we therefore decided for an AOSD approach and evaluated two AORE approaches for the first phase of the development cycle. We found that both of these approaches: Theme/Doc and MDSOCRE however fall short. Theme/Doc showed problems with requirements granularity and lack of aspectual interaction support. MDSOCRE presented a appropriate granularity, but however lacks an explicit way to express interactions.

To address these shortcomings, we developed extensions to both approaches that make Theme/Doc and MDSOCRE more suitable for the industrial problem at hand. The extensions made to Theme/Doc allow us to explicitly document the interaction between concerns as well as the requirements participating in the crosscutting and interaction relationships. The extensions made to MDSOCRE allow us to relate different requirements and explicitly support concern interactions.

It is our opinion that applying existing AORE approaches to industrial software is an important effort, as it may reveal shortcomings in these approaches, which is what we have shown here. Using these approaches in different settings will show avenues for improvement and extension of their applicability.

References

1. Recommended practice for software requirements specifications. *IEEE Std 830-1998*, 1998.
2. Elisa Baniassad and Siobhan Clarke. Finding aspects in requirements with theme/doc. In *Early Aspects Workshop at AOSD*, March 2004.
3. Elisa Baniassad and Siobhan 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. Elisa Baniassad, Paul C. Clements, Joao Araujo, Ana Moreira, Awais Rashid, and Bedir Tekinerdogan. Discovering early aspects. *IEEE Softw.*, 23(1):61–70, 2006.
5. David Bar-On and Shmuel Tyszberowicz. Derived requirements generation: The dras methodology. *Software Science, Technology and Engineering, IEEE International Conference on*, 0:116–126, 2007.
6. Siobhán Clarke and Elisa Baniassad. *Aspect-Oriented Analysis and Design. The Theme Approach*. Object Technology Series. Addison-Wesley, Boston, USA, 2005.
7. Gaming Laboratories International. *Gaming Devices in Casinos*, 2007. Available at: http://www.gaminglabs.com/.
8. 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.
9. Nevada Gaming Commission. *Technical StandardsForGamingDevices And On-Line Slot Systems*, 2008. Available at: http://gaming.nv.gov/stats_regs.htm.
10. Awais Rashid, Ana Moreira, and João 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.
11. Frans Sanen, Eddy Truyen, Bart De Win, Wouter Joosen, Neil Loughran, Geoff Coulson, Awais Rashid, Andronikos Nedos, Andrew Jackson, and Siobhan Clarke. Study on interaction issues. Technical Report AOSD-Europe Deliverable D44, AOSD-Europe-KUL-7, Katholieke Universiteit Leuven, 28 February 2006 2006.
12. Peri Tarr, Harold Ossher, William Harrison, and Jr. 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.
13. Arturo Zambrano, Johan Fabry, Guillermo Jacobson, and Silvia Gordillo. Expressing aspectual interactions in requirements engineering: experiences in the slot machine domain. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC 2010)*, pages 2161–2168. ACM Press, 2010.