Science of Computer Programming 78 (2012) 65–92

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

# Expressing aspectual interactions in requirements engineering: Experiences, problems and solutions

Arturo Zambrano [a,*], Johan Fabry [b,1], Silvia Gordillo [a,c]

[a] LIFIA, Facultad de Informática, Universidad Nacional de La Plata, La Plata, Argentina
[b] PLEIAD Laboratory, Computer Science Department (DCC), University of Chile, Santiago, Chile
[c] CIC, Pcia. de Buenos Aires, Argentina

ARTICLE INFO

*Article history:*
Received 5 January 2011
Received in revised form 7 December 2011
Accepted 8 December 2011
Available online 27 December 2011

*Keywords:*
Aspect oriented requirement engineering
Aspect dependences and interactions
Industrial case study

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 the limitations of both approaches we encountered and propose extensions that allow them to cope with concern interactions. We also show how these extensions provide the needed expressiveness by applying them to our industrial case study.



## 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 modeling of these crosscutting concerns as aspects, identifying and characterizing their influence on other requirements in the system [20,24]. 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 an as consistent as possible model of the system early in the software development life-cycle. As a result the system can be designed and built correctly, considering the needs of the various stakeholders.

In the context of an industrial project we have been re-implementing the software that runs on the casino gambling device best known as a slot machine (SM). 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 modularized 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. This was motivated by the argument of Liu et al. [19] that most feature interactions can be detected in early stages of the software development cycle, by reasoning on the causes of interactions and building models of them.

* Corresponding author. Tel.: +54 221 4228252.
*E-mail address:* arturo.zambrano@lifia.info.unlp.edu.ar (A. Zambrano).

[1] Johan Fabry is partially funded by the FONDECYT project 1090083. This article extends the article "Expressing aspectual interactions in requirements engineering: experiences in the slot machine domain" (Zambrano et al. 2010 [35]) that appeared in the Proceedings of the 2010 ACM Symposium on Applied Computing (SAC 2010). doi:10.1145/1774088.1774545.

0167-6423/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2011.12.004

The objective of this detection is to document 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 therefore be an, as consistent as possible, model of the requirements. As complete consistency is not possible in the presence of certain conflicts, in those cases we want to document them to defer their resolution to later development phases. Hence to accomplish this objective, it is necessary to be able to rely on expressive mechanisms in the selected modeling technique for this phase.

To the best of our knowledge there is however no previous work 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.

In the requirements engineering community, interactions between requirements are a well-known (intra) traceability problem [23]. In the AOSD community, and hence when applying AORE techniques, this problem maps to what is known as dependencies and interactions with aspects [8]. Dependency and interaction support is still an open issue for the AOSD community. Therefore, the focus of this work lies on assessing the support for expressing interactions between aspects or concerns, which, depending on the point of view, can be considered either a requirements traceability problem or a problem of dependencies and interactions with aspects.

We elected to perform requirements engineering using both the Theme/Doc approach [3] and Multidimensional Separation of Concerns for Requirements Engineering (MDSOCRE) [20], focusing on how these allow us to express and document aspectual dependencies and interactions. The choice of these two approaches was based on our perception of their maturity and of their acceptance in the AORE community, the latter as reflected by publication record. Due to the complexity of the domain and the time that the application of each approach requires, we were unable to perform the same experiment on other approaches such as AORA [6], AOSD/UC [18], or AOREC [17].

More concretely, Theme/Doc was selected because a book is available that describes at a detailed level its application to a arguably complex example (which also demonstrates some degree of scalability) [14]. Moreover, it also discusses later phases of development, for example how to deal with conflicts in design documents when using Theme/UML. MDSOCRE was selected because it claims to explicitly support conflict resolution and because of its flexibility due to the action/operator approach (discussed in Section 3.4.1) that allows us to express more than just crosscutting relationships.

Both of the approaches we evaluated enable us to partially 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 this paper, in addition to describing these drawbacks in more details, we also elaborate and validate extensions that address these issues. For example we propose new kind of relationships to Theme/Doc for documenting interactions, and add new interaction information in MDSOCRE.

We consider that this experience report is of use to the AORE community, as it evaluates representative approaches in an industrial setting and also proposes and validates enhancements to existing work. This experience is also useful for those who want to perform AORE in domains where interactions are present, to know which support is currently available and how the lack of some features may be addressed.

This paper is organized as follows: Section 2 is a brief introduction to the slot machine domain: it presents a decomposition of the application in different concerns and the interactions we need to model using AORE. The evaluated AORE approaches and the output the evaluation are presented in Section 3, including the limitations found for each approach. In Section 4 we propose extensions to the original approaches in order to cope with the limitations found. Section 5 describes a user study carried out in order to validate the expressiveness of the extensions performed to MDSOCRE. We present related work in Section 6 and Section 7 concludes. Two appendixes (Appendix A and Appendix B) contain the complete models we produced, showing the results of applying the extensions to our setting.

## 2. Concerns in the slot machine domain

A slot machine (SM) 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 while 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, and so on. One example of these are the Nevada Regulations [21].

**Standards** To ensure proper behavior of SMs, there are certification institutes that perform severals tests and quality checks on the SMs. The expected behavior of an SM is defined in documents called standards, for example the GLI standard [15].

**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 [16] (Game to Server) protocol, an open standard for communication of an SM with a back-end.

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 sizable set of documents using multiple terms for describing the same object, action or event. (We count approximately 150 pages of pure requirement specifications, while the complete documents that include clarifying text, examples and notes are approximately 2000 pages in size.) Furthermore, in some cases it is necessary to complement and normalize different sources referring to the same topic. For instance, consider the case of *Error Conditions*, which are treated by both the Nevada regulations [21] and the GLI standard [15]; some of the conditions specified by the regulations match, but others are defined by only one of them. Note that the final SM, if needs to be installed in Nevada, must comply with the error conditions defined in both documents.

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.

### 2.1. Concern organization

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 organize 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.

**Meters** 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, and so on.

**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, and so on. For our work, we consider either using G2S or a proprietary protocol. In the remainder of this text we will refer to the generic *communication* concern or, when appropriate, we will specify which protocol we are considering.

**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 work, simulating events such as entering money or winning a prize.[2] Note that any meter or data changed due to operation in demo mode should not be persisted or reported, as it is simulated behavior.

As the domain is complex, there may be other possible concern decompositions. We choose this one because, based on our experience, it properly modularizes 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 implementations or compositions of components or aspects to comply with the regulations of each scenario.

### 2.2. Selected requirements

Due to the large number of requirements of our case (around 600), we only show here a small subset of requirements that illustrate the main concerns for the SM domain and the four important types of interactions discussed later: conflict, dependency, mutex and reinforcement.

The requirements we present here are a slightly simplified and summarized version taken from the different original sources (regulations, standards and protocol specifications). These documents are produced by the corresponding organizations: certification laboratories, protocol steering committees, and institutions that regulate the gaming industry. It is important to note the particular nature of these requirements: as these are legal requirements, **all** of them must be fulfilled in the resulting application. Moreover, all of them are known and present at the start of the development process. Part of the formal certification process of the software that is installed in the SMs is ensuring that all these legal requirements are met. Also, the requirements we discuss here change at a slow pace (which can be measured in years) when compared with

[2] SMs present in casinos do not have the demo feature active, obviously.

**Table 1**
Requirements for SM domain concerns 2.

| | | | |
|---|---|---|---|
| Game | | | |
| R-SM-1 | Slot machines have 5 reels. | R-SM-4 | A slot machine has one or more devices for entering money. |
| R-SM-2 | Reels spin when play button is pressed. | R-SM-5 | As money is inserted credits are "assigned" to the player. |
| R-SM-3 | Prizes are awarded according to a pay table. | R-SM-6 | A slot machine must provide some means for cashing the credits out. It could be a ticket printer, a coin hopper. |
| Game recall | | | |
| R-GR-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. | R-GR-2 | Last play information shall provide all information 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. |
| Program interruption and resumption | | | |
| R-PR-1 | After a program interruption (e.g. processor reset), the software shall be able to recover to the state it was in immediately prior to the interruption occurring. | R-PR-3 | An SM must store all meter information in persistent memory. |
| R-PR-2 | Restoring Power. If a gaming device is powered down while in an error condition, then upon restoring power, the specific error message shall still be displayed and the gaming device shall remain locked-up [. . .]. | | |
| Demo mode | | | |
| R-DM-1 | The Slot Machine must permit a test, diagnostic or demo mode, which permits a gaming device (*e.g.* a hopper) to be tested. The test shall be completed on resumption of normal operation. | R-DM-5 | Specific meters are permissible for these types of modes provided the meters indicate as such. |
| R-DM-2 | If the gaming device is in a test, diagnostic or demo mode, any test that incorporates credits entering or leaving the gaming device (e.g., a hopper test) shall be completed on resumption of normal operation. | R-DM-6 | Test/diagnostics mode may be entered, via an appropriate instruction, from an attendant during an audit mode access. These modes should not be accessible to the player. |
| R-DM-3 | There shall not be any mode other than normal operation (ready for play) that increments any of the electronic meters. | R-DM-7 | When exiting from test-diagnostic mode, the game shall return to the original state it was in when the test mode was entered. |
| R-DM-4 | Any credits on the gaming device that were added during the test, diagnostic or demo mode shall be automatically cleared before the mode is exited. | R-DM-8 | Test Games. If the device is in a game test mode, the machine shall clearly indicate that it is in a test mode, not normal play. |

the time taken by the development of a SM game. This is due to the nature of the source organizations that produce the legal requirement documents.

We have grouped the requirements according to the concerns presented in Section 2.1 to favor the understanding of the different interactions which are the core of this work. In Tables 1 and 2 we summarize some requirements by concern, grouped by concern and not by the document where they are defined. For a complete listing of them we refer to the various requirements documents [15,16,21].

### 2.3. 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 [28]. In this report, the authors classify aspect interactions into dependency, conflict, mutex and reinforcement, as follows:

**Dependency** In this case one aspect explicitly needs another to work correctly.
**Conflict** 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.

**Table 2**
Requirements for SM domain concerns.

| | | | |
|---|---|---|---|
| Meters | | | |
| R-M-1 | Credit meter: shall at all times indicate all credits or cash available for the player to wager or cashout. | R-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 initialization. |
| R-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, bills, Ticket/Vouchers, coupons or other approved notes accepted. | R-M-4 | Meters should be updated upon occurrence of any event that must be counted, including: play, cashout, bill in, coin in. |
| R-M-5 | G2S meters are: *gamesSinceInitCn* Number of games since initialization. *WonCnt*: Number of primary games won by the player. *LostCnt*: Number of primary games lost by the player. | | |
| Communications: G2S | | | |
| R-G2S-1 | The G2S protocol is designed to communicate information between an SM, and one or more host systems. | R-G2S-4 | The device can generate an event in a unsolicited manner or in response to a host command |
| R-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 at predetermined intervals. | R-G2S-5 | Current timestamp can be set by the host. |
| R-G2S-3 | Information provided by the SM is used for audit purposes. | R-G2S-6 | Command GetGameRecallLog is used by a host to request the contents of a transaction log of last plays from a SM. |
| Communications: proprietary comm. protocol (PCP) | | | |
| R-PCP-1 | The PCP communicates a SM with a host system. | R-PCP-2 | The PCP must report meters of a SM when required by the host. |
| R-PCP-3 | Configuration settings such as current timestamp are configured from the host. | | |
| Error conditions | | | |
| R-EC-1 | Gaming devices shall be capable of detecting and displaying error conditions and illuminate the tower light for each or sound an audible alarm. | R-EC-5 | Error conditions shall be communicated to an on-line monitoring and control system when this is available. |
| R-EC-2 | Error conditions should cause the gaming device to lock up and require attendant intervention. Error conditions shall be cleared either by an attendant or upon initiation of a new play sequence after the error has cleared except for those deemed as a critical error. | R-EC-6 | An event represents an occurrence of an incident detected by a device in an EGM. |
| R-EC-3 | Error conditions are: coin jam, reverse coin in, stacker full, bill jam, external doors open. | R-EC-7 | Important events must be reported in real-time, including: error conditions, tickets inserted, ticket printed. |
| R-EC-4 | Video based games shall display meaningful text as to the error conditions. | | |

**Mutex** A mutual exclusion occurs when two aspects implement the same functionality, but only one of them can be used at a time.

**Reinforcement** This is a positive interaction where an aspect influences the correct working of another, allowing it to provide extended functionality.

Based on our previous experience implementing software for SM, we observed the existence of these aspect interactions between the concerns identified in our domain. For example, 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 present 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.

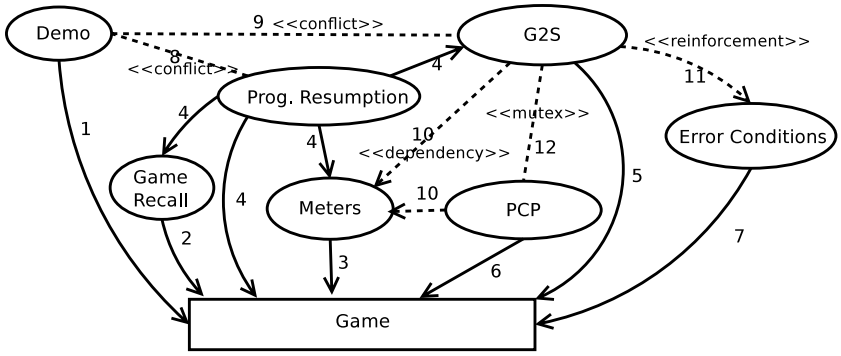


**Fig. 1.** Concern interactions in an ad-hoc notation. Regular arrows indicate crosscutting, dashed arrows indicate interactions between concerns, tagged with UML-like stereotypes.

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 concerns (or some requirements of them) 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. A representative selection of these is shown in Fig. 1, which uses ad-hoc notation where *Game* is the base concern (represented by a square) and crosscutting concerns are represented by ovals. Different arrows are used to indicate crosscutting and interaction relationships. More in detail, 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 behavior for testing purposes.
2. *Game Recall* to *Game*: *Game Recall* requirements affect many aspects of the *Game*'s behavior, 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. Note that it also crosscuts PCP, but we did not include this here to avoid clutter.
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 the reporting system.
6. *PCP* to *Game*: This refers to another protocol used for the same purpose as G2S, that is to monitor the game's behavior.
7. *Error Conditions* to *Game*: The behavior associated to error conditions need to be woven into the game behavior. Requirements in *Game* that could raise an error condition vary: from a bill inserted to a door opened.

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*: 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 *PCP* on *Meters*: Some data reported to the RS is stored or can be derived from meters. Hence, communication protocols need the meters to be up to date in order to accomplish their purpose.
11. **Reinforcement** of *G2S* with *Error Conditions*: As we mentioned in Section 2, some parts of the G2S protocol are not mandatory for specific instances. When error conditions are tracked in the game, additional behavior is made available in G2S, such as real time event reporting.
12. **Mutex** between *G2S* and *PCP*: 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.

As said above, this is only a selection of some representative interactions. There are many more that we do not include in Fig. 1 to avoid cluttering of the graphic. For example: G2S and PCP depend on GameRecall as both protocols require to retrieve information of the last plays, and Program Resumption crosscuts PCP, for the same reason as it crosscuts G2S.

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 section we discuss Theme/Doc [3] and Multidimensional Separation of Concerns for Requirements Engineering [20], showing if and how they make it possible to express and document aspectual interactions.

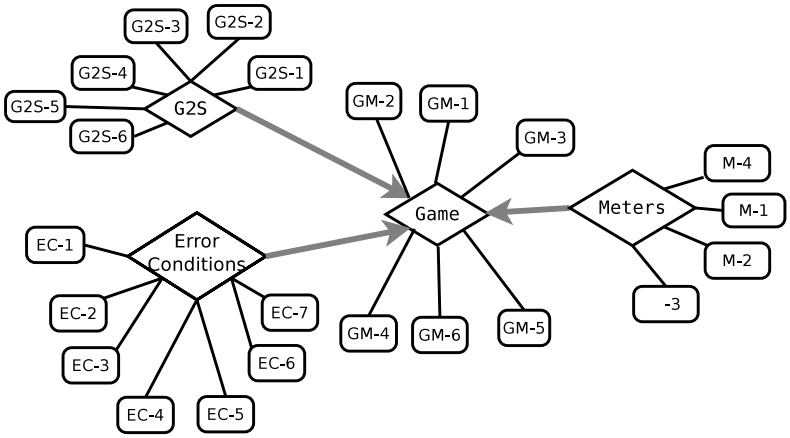


**Fig. 2.** *Game*, *Meters*, *G2S* and *Error Conditions* concerns expressed using the Theme/Doc notation.

## 3. Evaluation of AORE approaches

### *3.1. 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 [3,14], which also treats aspectual design (Theme/UML).

We selected Theme/Doc because it explicitly supports passing information from the requirements analysis to the design phase, which could include interactions. Besides this, the Theme book [14] mentions conflict resolution as a feature of the Theme approach, which is one interaction type we are interested in. Also, in terms of scalability, it was applied to a non-trivial example in the book, which makes it an interesting candidate for evaluation.

#### *3.1.1. Brief overview of Theme/Doc*

Theme/Doc [2] is the requirement analysis part of the Theme approach [3,14]. In Theme/Doc, requirements are organized 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, and so on.

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 Section 2.1 as themes. We will detail our motivation for this in Sections 3.3.3 and 3.3.4.

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 [14]:

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 behavior described by the dominant theme is fired as a result of the execution of some behavior from the base theme.
4. The triggered theme is fired in multiple situations: the crosscutting behavior must be executed in several cases, not just one.

An important feature of Theme/Doc is its visual support through diagrams, which helps in the understanding of the requirements model the system to be. In Theme/Doc views, requirements are represented by rounded boxes, and they are organized 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.

The Theme book [14] claims that Theme/Doc has tool support through a web application.[3] Unfortunately, this software was not available at the time of our experiments. To the best of our knowledge, there are no other tools with direct support for Theme/Doc. We therefore followed the notation from the book to build our diagrams by hand.

[3] Stated to be available at: http://www.thethemeapproach.com:8081.

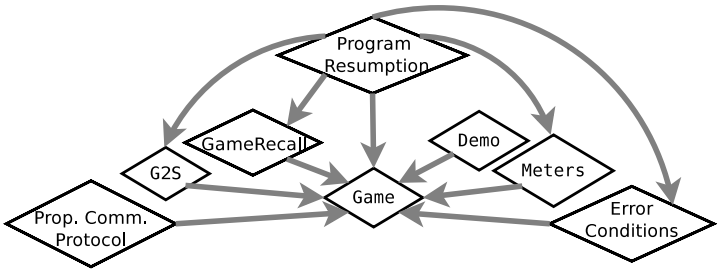


**Fig. 3.** Crosscutting relationships between themes using the Theme/Doc graphical notation.

### 3.2. Use 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 correctly establish 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.

Fig. 3 shows crosscutting among the themes presented in Section 2.3. For clarity, here we just present the crosscutting relationships between the themes and without including the requirements. The diagrams showing all the above concerns with their more significant requirements are included in Appendix A. Note that in contrast with Fig. 1 (which uses an ad-hoc notation for showing crosscutting and interaction relationships) in this case only the crosscutting information is present due to the limitations of Theme/Doc, which are analyzed in the following sections.

### 3.3. Limitations of Theme/Doc

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

#### 3.3.1. 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 – in the identification of crosscutting themes – of Theme/Doc, but it is not made explicit.

#### 3.3.2. 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 these interactions 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 [30]. 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.

#### 3.3.3. Implicit requirements

As a consequence of performing a domain knowledge based analysis of our requirements and concerns we also were confronted with the fact 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*.

For example, during interaction analysis it is possible that new requirements arise because of interactions that need to be resolved. Consider the following example:

> G2S provides time and date configuration for slot machines. A proprietary protocol also provides the same configuration. A SM can be connected using both protocols at the same time.

In this case there is an implicit mutex relationship present. The machine can take the time and date from any of the protocols, but not from both of them at the same time. This would lead to erratic behavior in case that both times do not exactly match. Hence, the following issue arises: Which time and date source should the SM take when both protocols are active? A decision must be made in the requirements analysis, for example stating that G2S is preferable over any other source of time and date. This decision helps to resolve the interaction problem by defining part of the behavior of the system, and must be recorded.

#### 3.3.4. Ambiguity of the requirements

An ideal requirements specification should 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 first results in synonyms being used in different documents. Second, and more importantly, 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, we consider it 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 however the requirement and the theme could be related by implicit actions, as outlined above. Furthermore, the same issue is present 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 [14]. 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. Put differently, the name-based approach proposed originally by Theme does not work in our setting due to the nature of the requirements sources. Having several large documents which present not only words but key ideas using a different vocabulary prevents the use of the name-based approach of Theme/Doc.

Considering the kind of requirements we face, we consider two options to resolve ambiguities. The first one is to rewrite all the requirements, normalizing them to use the same vocabulary; the second one is to use domain knowledge to associate requirements to the corresponding themes directly. Due to the large number of requirements (approximately 600) and presence of multiple sources the first option is not feasible, we therefore opt for the second. Note that grouping requirements into concerns based on domain knowledge is not new [20,4]. Our experience is that the resulting concerns are useful as they can be easily discussed with domain experts.

### 3.4. MDSOCRE

MDSOCRE (Multidimensional Separation of Concerns in Requirements Engineering) is the evolution of a line of AORE approaches such as PreView [29] and ARCaDe [25]. As it arguably provides the most expressive and flexible constructs for binding crosscutting concerns to base concerns, we chose it as the second case in our study.

#### 3.4.1. Brief overview of MDSOCRE

Multidimensional Separation of Concerns in Requirement Engineering (MDSOCRE) [20], is a refinement of the ARCaDe approach [25]. 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 analyze crosscutting behavior. In contrast to Theme, it does not provide a graphic notation, instead it uses XML to express requirements and composition rules.

MDSOCRE aims to eliminate conflicts, which are the result of contradictory concerns. These 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 Section 2.3 (taken from the AOSD-Europe study of interaction issues [28]). 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 run-time 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 Section 3.6.

MDSOCRE is supported by the EA-Miner tool [26]. This tool is an Eclipse plugin[4] that needs the requirements to be entered in plain text format. As part of its processing, EA-Miner uses a web service to parse natural language. Unfortunately, this service was not working at the time of our experiments, and we were not able to use the tool.

[4] Downloadable from http://www.aosd-europe.net/deliverables/d108EAMinerVersion2.zip.

### 3.5. Use of MDSOCRE

We were able to build a complete requirements model of the SM application using MDSOCRE. Listing 1 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 sub-requirements. We do not include the detailed listing of all concerns here and instead refer to Appendix B.

**Listing 1**. Game and Meter concerns expressed using MDSOCRE.

```
<Concern name="Game">
  <Requirement id="1"> A slot machines have 5 reels.
  </Requirement>
  <Requirement id="2"> Reels spin when 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. It
      could be a ticket printer, a coin hopper.
</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  (GLI 11 4.10.1)
  </Requirement>
  <Requirement id="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, bills, Ticket/Vouchers, coupons or
       other approved notes accepted. (GLI 11 4.10.5)
  </Requirement>
  <Requirement id="3"> Accounting Meters (GLI 11 4.10.9):  Coin In: a meter that accumulates
      the total value of all wagers [...]. Games-played: accumulates
            ....
   </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>
```

*Composition rules* are used to express crosscutting relationships. Listing 2 shows composition rules, 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. The operators express temporal intervals, temporal points or restrictions between sets of concerns. The outcome tags (*satisfied* and *fulfilled*) define the result of a composition. *Fulfilled* is used to denote that composition constraints have been imposed. *Satisfied* takes other requirement IDs as parameters and indicates that those requirements have been satisfied as a consequence of the imposed constraint [20].

**Listing 2**. A composition rule for Meters and Game using MDSOCRE.

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

**Table 3**
Expressing interaction types in MDSOCRE.

| | Action | Operator |
|---|---|---|
| Dependency | Ensure | with |
| Reinforcement | Provide | for |
| Mutex | Enforce | xor |
| Conflict | Enforce | xor |

The first composition rule of 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. Otherwise the action value should have been "satisfied" and the set of requirements that are satisfied. This is the case of the *Error Condition* composition, because when such a condition is detected an action must be taken, *i.e.,* an additional 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 parameterized `constraint` tag helps to express different variants of crosscutting relationships even though MDSOCRE does not natively support this. We were able to combine actions and operators to document the interactions, as shown in Table 3.

We used the action *ensure* and the operator *with* to represent a *Dependency* interaction. This follows the informal definition by Moreira et al. [20], 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. For *Mutex* the combination is the action *enforce*, to impose additional conditions with the operator *xor*, as we want to prevent the simultaneous activation of two implementations of the same functionality. Finally, note that for *Conflict* we used the same combination, since operationally the desired effect is to satisfy one just one of the requirements sets involved (similar to *Mutex*). The downside of using this notation is that both interactions are documented in the same way, even though their semantics is clearly different.

### 3.6. Limitations of MDSOCRE

Although we were able to completely model the SM application, we were faced with two limitations of MDSOCRE which resulted in models that are sub-optimal. Firstly there is no explicit interaction relationship, and secondly there is no support for unifying disparate requirements. We discuss these limitations next.

#### 3.6.1. Lack of interaction relationships

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 Section 2.3, interactions occur even between concerns without a crosscutting relationship. In our case we need to express 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: error events. These interactions as well as mutex (see Section 2.3) are not explicitly supported by this approach.

As a workaround we have combined pairs of actions and operators, as shown in Table 3. This solution however has three 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 . . ." [20].
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.
3. Using the semantics provided for actions and operators we could not find a way for expressing *conflicts*. This is probably due to the approach being focused on removing conflicts, hence no conflicts should need to be documented.

We consider the second downside to be a key factor. The mapping from Table 3 needs to be used to interpret implicit information that instead should be explicit. This makes such interpretation in this approach error prone, as we will show in Section 5.

An alternative would be the use of meta concerns, which seem to be a natural place to store information regarding interactions. Extending MDSOCRE in this way implies adding information regarding the interaction type and the interacting concern. This could be done in a number of ways using XML, for example by adding a tag parameterized with two attributes: the interaction type and the name of the interacting concern. Meta concerns are not exactly aimed at this purpose, but with this small 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.

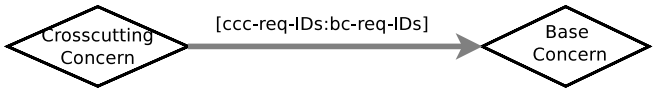


**Fig. 4.** Quantification label applied to a crosscutting relationship.

*3.6.2. No support for unification*

As in Theme, the ambiguity of requirements in the slot machine domain again impacts the process. Recall that in Section 3.3.4 we discussed the impact of having multiple and ambiguous requirement documents. For example, we may have different documents that list the meters or counters that must be provided by the SM, and furthermore in one document meters can be defined in multiple requirements. In other words the requirements of meters are scattered over multiple documents. It is however necessary to group all this information at design time to correctly design the meters concern.

One possibility here is to rewrite requirements so that meters listing is done just once, containing meter definition from all the sources. However such a rewriting effort is only feasible when considering a small number of requirements. Furthermore, a downside of creating a unified list is that after requirements fusion it is hard to update these once (one of) the sources evolves.

Alternatively, we can keep requirements organized as in the originating document, removing the need for unification as well as the issue of requirements changes. However, it is desirable to have some kind of link between them, as these different requirements complement each other, *e.g.* they jointly form the list of all meters. This allows the engineer to analyze all the requirements defining the same concept. XML permits doing this as it is possible to add cross-references between different elements in the tree. However MDSOCRE lacks a facility for describing such cross-references.

## 4. Extensions of the existing approaches

### *4.1. Extensions to Theme/Doc: Theme/Doc-i*

We now present some enhancements to Theme/Doc and give examples in the SM domain. These allow us to deal with the issues expressed in the previous section, as we will demonstrate in Section 4.1.4.

*4.1.1. Granularity*

In order to improve the information of requirements implicated in a crosscutting relationship, we added *quantification labels* to the existing gray arrows of Theme/Doc. Quantification labels allow us to specify which requirements are involved in a given crosscutting relationship, from both sides: the crosscutting concern and the base concern.

As shown in Fig. 4, a quantification label has two parts separated by a colon:

**Crosscutting requirements IDs** this 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 a list, a range, or the keyword *all* that indicates which requirements are the requirements affected by the crosscutting (the destination of the arrow).

Before deciding in favor of quantification labels, we evaluated the use of graphical elements to indicate affected requirements. We considered, for example, adding arrows from affecting to affected requirements. We discarded such graphical approaches as we found that they add a large amount of clutter to the diagrams, making comprehension more difficult.

In summary, quantification labels allows us to express fine grained information that otherwise would be missed using the original Theme/Doc. This information can be used in later phases to take special care of the design and the implementation of the requirements involved in such relationships. For example, quantification labels used in the *base* side of a crosscutting relationship can help in the definition of pointcuts during design.

*4.1.2. Expressing interactions*

As outlined in Section 3.3.2, we also need to model interactions between themes, since multiple themes need to be combined to form a system. As interactions are a new type of relationship between concerns, we decided for a notation that is consistent with the Theme/Doc notation that relates concerns. We propose to document interactions using dashed lines or dashed arrows, depending on whether the interaction is directional or not. To this line or arrow a text label is attached, indicating which type of interaction is represented (mutex, conflict, reinforcement or dependency).

The choice of a line or an arrow depends on the type of the interaction. For some interactions, such as *mutex* and *conflict*, there is no direction, since they are symmetric relationships, hence no arrow is needed. However, in the case of *reinforcement* and *dependency*, it is important to specify the direction of the relationship in order to understand which concern is reinforcing the other concern, or which concern depends on the other. Hence for those interactions arrows are used. The different kinds of interaction notations are shown in Fig. 5.

Interaction arrows can be combined with quantification labels in order to document which specific requirements are involved in the relation. Examples of these combinations are shown in Section 4.1.4.

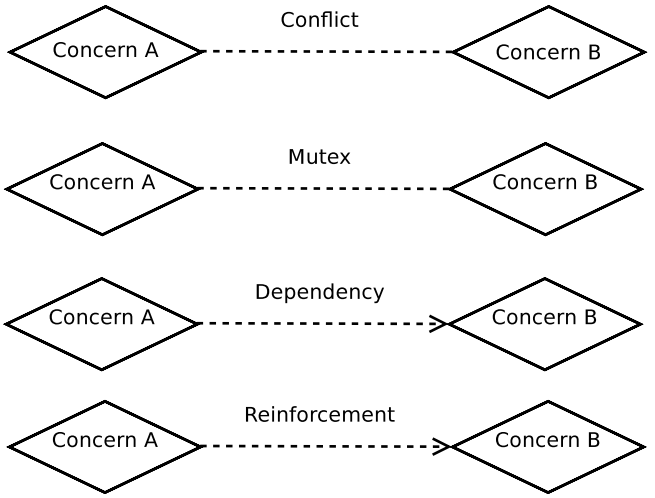


**Fig. 5.** Interaction relationships.

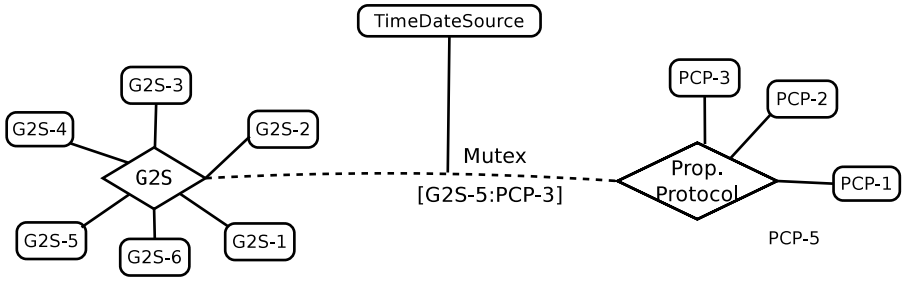


**Fig. 6.** A new requirement related to a mutex interaction.

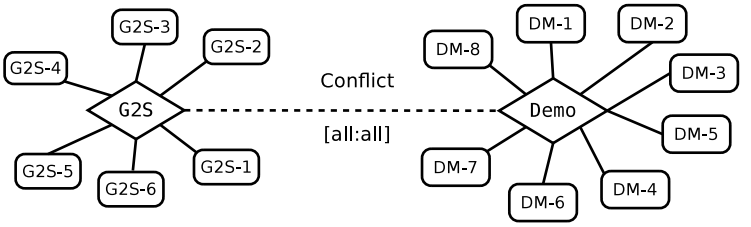


**Fig. 7.** Conflict between Demo and G2S concerns.

*4.1.3. Implicit requirements, ambiguity of requirements*

Both issues of implicit requirements and the ambiguity of requirements benefit from the intervention of a domain expert. On the one hand this will make explicit the requirements that are implicit, and on the other hand this allows the requirements documents to be disambiguated. This implies adding some specific task in the process of Theme, probably during the requirements processing (where requirements are split, removed or added).

Considering the example of the implicit, derived requirement we presented in Section 3.3.3, we propose to add such requirements to the affected interaction itself. Fig. 6 shows how a derived time data source requirement is attached to the mutex interaction.

*4.1.4. Theme/Doc-i applied*

Using our extensions, we were able to explicitly specify all the interactions in the case study. In this section we provide examples for the different types of interactions we have modeled. We refer to Appendix A for the full requirements model.

In Fig. 6 we have seen how a mutex interaction is represented. Fig. 7 shows how a conflict (explained in Section 2.3) is documented using the proposed extensions. Note that besides the interaction type, the requirements involved are specified using quantification labels.

Fig. 8 shows a reinforcement between *Error Conditions* and *G2S*. In this case, it is important to specify the direction of the relationship, since it should be clear which concern is reinforcing which other concern. Again, quantification labels have

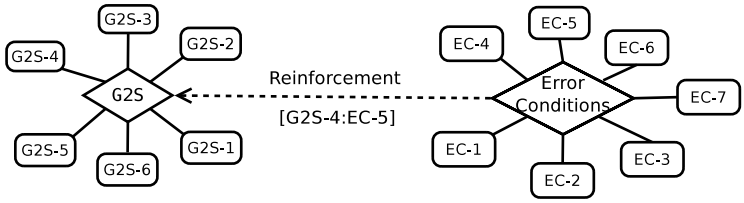


**Fig. 8.** *G2S* concern reinforced by *Error Conditions* concern.

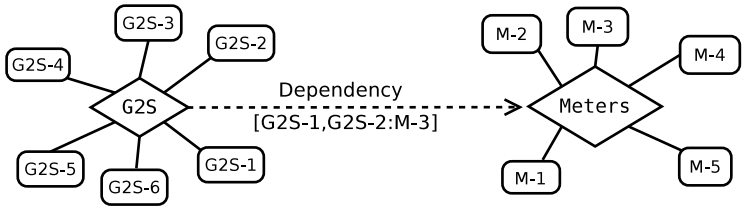


**Fig. 9.** *G2S* concern depends on *Meters* concern.

been used to identify requirements. From Fig. 8 it is clear that requirement 5 of *Error Conditions* enables extra behavior in the *G2S* concern, specifically it enables requirement 4, (*Devices can generate an event in unsolicited manner . . .*).

*G2S* (and any protocol that needs to report the SM state) depends on *Meters* to provide the required information. This situation is documented in Fig. 9, which shows that in order to satisfy requirement 2 of *G2S* first requirements 4 and 5 from the *Meters* concern need to be satisfied.

### *4.2. Extensions to MDSOCRE: MDSOCRE-i*

Considering the shortcomings of MDSOCRE outlined above, we propose an extension to MDSOCRE, called MDSOCRE-i. It consists of two parts: explicit interaction relations to address the need for clear notation of interactions, and cross-references to allow unification of scattered requirements into one complete set of requirements for a concept.

#### *4.2.1. Explicit interaction relations*

Unfortunately, MDSOCRE does not natively support the notion of interactions. Although they can be expressed using a combination of actions and operators, as in Table 3, we do not consider this a clean solution. This is because interactions are not documented in a way that makes it easy to differentiate the different types of interactions (as confirmed by experimental results in Section 5). Note that in our requirements engineering we were forced to disregard parts of the MDSOCRE methodology: originally conflicts are not supposed to be expressed in the resulting models as they should have been removed as part of the requirements engineering process. Instead we need these conflicts to be explicitly present.

An alternative to our workaround is the use of the meta concern facility to store information regarding interactions. However there is a conceptual mismatch here as it means storing concrete information in an artifact that is aimed at expressing generic information regarding concerns.

We assert that a new relationship between concerns, aimed at documenting interactions, is needed. This new relationship would enable us to express interactions between concrete concerns, as well as between meta concerns when necessary. We propose an extension for MDSOCRE to effectively document interactions and call this extension MDSOCRE-i.

Before deciding for the extension presented here we evaluated different solutions. For example, having new actions or operators was considered as a possible solution. We however discarded this because some combinations of existing actions and operators with new actions or operators would not make any sense. Another possibility we evaluated was to replace the `Composition` XML tag by an `Interaction` tag, which would be a much more intrusive change to the MDSOCRE specification. Moreover, interactions form part of the information of a composition, and therefore this option was also discarded.

We propose to extend the `Composition` XML element of MDSOCRE, allowing it not only to express crosscutting relationships but also interactions. In listing 3, we show a new `Interaction` XML element that can be parameterized with the interaction type. We do not pose any restriction on this type, and here simply specify the interaction types as used in the previous sections of the text. Note that we only apply this extension at the base level, but it can also be extended to apply at the meta concern level.

The interaction element is always contained in a requirement element and nests a second requirement element. The containing requirement element affects the nested requirement element as specified by the type of the interaction element. In other words, the direction of the interaction relationship (when relevant) is from the outer requirement to the inner one. Consider for example Listing 3 that describes the dependency between G2S (requirement 6) and Game Recall (requirements 1 and 2). As we explained in Section 2.3, *G2S* depends on *Game Recall*, as it needs the information captured by *Game Recall* in order to communicate it to the on-line monitoring systems when needed.

**Listing 3**. The `Interaction` element instantiated for a *dependency*

```
<Composition>
 <Requirement concern="G2S" id="6">
  <Interaction type="dependency">
   <Requirement concern="GameRecall" id="1,2"/>
  </Interaction>
 </Requirement>
</Composition>
```

Note that the `Requirement` tag has not been modified with respect to the original MDSOCRE specification as it already allows to specify which requirements participate in a composition.

Listing 4 shows examples for the other three interactions (reinforcement, mutex and conflict).

**Listing 4**. Reinforcement, conflict and mutex interactions

```
<Composition>
 <Requirement concern="Meters" id="2,3">
   <Interaction type="reinforcement" >
     <Requirement concern="Proprietary Communication Protocol" id="4"/>
  </Interaction>
 </Requirement>
</Composition>

<Composition>
 <Requirement concern="Demo Mode" id="1,4">
    <Interaction type="conflicts" >
      <Requirement concern="Game Recall" id="1,2"/>
    </Interaction>
 </Requirement>
</Composition>

<Composition>
 <Requirement concern="Demo Mode" id="9">
    <Interaction type="mutualExclusion" >
      <Requirement concern="Game" id="3"/>
    </Interaction>
 </Requirement>
</Composition>
```

Note that in the case of mutex and conflict interactions the relationships are symmetrical, so the order of the requirements may vary with no impact: It is equivalent to say that Game Recall conflicts with Demo or Demo conflicts with Game Recall. In the case of mutex, the symmetrical nature of the relationship is more obvious as it is used to document two (or more) instances of a given functionality. For example, Listing 4 documents the mutual exclusion between two requirements, one of the Game concern and one of Demo concern. The mutual exclusion is required because both of them determine a different way of generating the outcome for a play.

#### *4.2.2. Cross-references for unification*

As indicated in Section 3.6.2, in the SM domain requirements coming from different sources describe the same concepts. These different sources need to be unified to eliminate ambiguities and to ensure that the requirements model is complete. This is because requirements may complement each other while originating from different documents, or from different parts of the same document. Furthermore changes in a requirement may impact all the requirements that it complements, in the same document as well as in other documents. To allow such unification and a more straightforward assessment of the impact of changing a requirement, MDSOCRE-i contains a second extension: adding cross-references to a requirement.

The cross-reference extension is meant to be used inside of a single concern to link requirements describing different perspectives of the same concept. Our extension consists of a `seeAlso` attribute that lists the identifiers of related requirements, *i.e.,* specifying a cross-reference. When adding complementary requirements to an existing set of requirements, the requirement engineer adds this attribute to the requirement tags. This attribute is added both in the existing set of requirements, referring to the newly added requirement that complements them, as well as to the new requirement, referencing the old requirements that are being complemented.

In Listing 5 we show an example of how a requirement defining some meters, taken from the G2S documentation refers (using the `seeAlso` attribute) to requirements 1, 3 and 4 of the same concern, defined in the GLI documentation that contain complementary information.

**Table 4**
Limitation of Theme/Doc and proposed extensions.

| Limitation in Theme/Doc | Extension in Theme/Doc-i |
|---|---|
| Granularity (Section 3.3.1) | Quantification labels (Section 4.1.1) |
| Expressing interactions (Section 3.3.2) | New arrow notation combined with quantification labels (Section 4.1.2) |
| Ambiguous and implicit requirements (Sections 3.3.3 and 3.3.4) | Disambiguation, derived requirements (Section 4.1.3) |

**Table 5**
Limitation of MDSOCRE and proposed extensions.

| Limitation in MDSOCRE | Extension in MDSOCRE-i |
|---|---|
| Lack of explicit interaction relationships (Section 3.6.1) | New tags and attributes to describe interacting concerns and requirements (Section 4.2.1) |
| No support for unification (Section 3.6.2) | New tag and attribute to document relationships between requirements describing closely related topics. (Section 4.2.2) |

**Listing 5**. See also extension

```
<Concern name="Meters">
<!-- From GLI 11 -->
 <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 (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, bills, Ticket/Vouchers, coupons or other
     approved notes accepted.
 </Requirement>
 <Requirement id="3" seeAlso="1,4,5"> 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.
 </Requirement>
 <Requirement id="4" seeAlso="1,3,5"> Meters should be updated upon occurrence of any event
     that must be counted, including: play, cashout, bill in, coin in.
 </Requirement>
<!-- From G2S Docs-->
 <Requirement id="5" seeAlso="1,3,4"> Some 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.
 </Requirement>
</Concern>
```

#### *4.2.3. Applying MDSOCRE-i*

We successfully used MDSOCRE-i to build a complete requirements model of the SM application. We were able to model all interactions present in the requirements, and to use cross-references to build complete sets of requirements for all concepts that are defined in a scattered form. Considering the size of the specification documents, we found that MDSOCRE-i yielded slightly shorter documents: 228 lines versus 236 lines for MDSOCRE. This even though extra cross-reference information is available. The full model is included in Appendix B.

### *4.3. Summary of extensions*

In this section we have proposed a number of extensions to both Theme/Doc and MDSOCRE to address the weaknesses we encountered when performing requirements analysis. The Tables 4 and 5 summarize the weaknesses of the approaches we studied and the extensions we made to address them.

The extensions to Theme/Doc we propose allow us to express information that otherwise would not be present in the requirements specifications we produce. Our extensions to MDSOCRE that yield MDSOCRE-i however do not add new expressive capabilities for interactions to the methodology. Instead they aim at helping the requirements engineer to easily understand interactions as contained in the requirements specifications we produce. In order to validate this hypothesis, we performed a user study that is presented next.

## 5. User study: MDSOCRE, MDSOCRE* and MDSOCRE-i

As we have discussed above, both MDSOCRE and MDSOCRE-i are sufficient to be able to model interactions thanks to the use of the mappings specified in Table 3. We call the variant of MDSOCRE that includes these mappings MDSOCRE*.

The aim of MDSOCRE-i however goes beyond MDSOCRE*: it also aims to achieve a conceptually cleaner fashion to model interactions, leading to a modeling phase that is faster and a model that is more complete and less error-prone to interpret. We have performed an additional comparison between MDSOCRE (including MDSOCRE*) and MDSOCRE-i to verify whether this is indeed the case, in the form of a restricted user study.

In our user study we compared interaction support of the two approaches in terms of accuracy and speed by means of two experiments. In other words, we compared legibility of the explicitly labeled interactions versus the action–operator pairs we used in Section 3.6.1. In the first experiment we established if the action–operator pair notation has legibility issues, by evaluating the advantages of using Table 3 (*i.e.,* using MDSOCRE*) in the requirements interpretation process. We found that the use of Table 3 significantly improves legibility of the requirements model. In the second experiment, we therefore compared MDSOCRE-i against MDSOCRE*, to see if MDSOCRE-i outperforms this setup. We found that indeed even in this case MDSOCRE-i is a significant improvement.

In each of the two experiments we tested legibility of the four interaction types as follows: In a first phase we established legibility of the first notation, and in a second phase of the second notation. Each phase consisted of four tasks, covering the four interaction types. In each task two concerns were presented with their interaction described in the corresponding notation. Then, a multiple choice of three different interactions between the two concerns was presented. The subject had to select which of the options was correct, yielding a measure of accuracy. The time needed to solve each task was also measured, in order to establish which approach delivers faster results while working with interactions. For both experiments all the subjects were in the same room. A single clock was available so that the subjects could take note of the current time for each individually finished task. The final times taken for each task was calculated by taking the difference between the time of the current and the previous task. Lastly, in order to evaluate the subjective preference for each approach, after executing each experiment a survey was completed by the subjects. It inquired as to their preference of notation per interaction type, graded on a five-point Likert scale.

The details of each case study are described in the following sections.

### 5.1. Case study 1: MDSOCRE versus MDSOCRE*

The first experiment we performed aimed to evaluate the original MDSOCRE, without knowledge of the semantics of the operator–action pair notation given in Table 3, versus MDSOCRE where this semantics is explicitly defined, which we call MDSOCRE*. Note that MDSOCRE* does not make any modifications to the MDSOCRE notation, the only difference is the semantics that is explicitly defined in Table 3.

*Evaluators.* The subjects of the study were 7 experienced IT professionals, but not knowledgeable in the SM domain. Each of them at least worked for 4 years using requirements, most of them in the role of requirements analyst. The group has an average experience of 11.5 years working on commercial IT projects.

*Activities.* The requirement interpretation tasks presented to the subjects were not taken from the SM domain, to avoid confusion because of unknown terminology. Instead, generic requirements were generated for their interpretation. For example, Listing 6 shows the requirement specification testing *Dependency*, in the second phase of the experiment treating MDSOCRE*. The multiple-choice options given to the subject for this task were the following (the correct answer is the third option):

1. User history features can be used without the availability of storage facilities.
2. User history benefits from the availability of storage facilities when available.
3. User history features depend on the availability of storage facilities.

**Listing 6**. Task evaluating Dependency in MDSOCRE*

```
<Concern name="Context-awareness">
 <Requirement id="1">Context-monitoring is the repeated observation of an entity's context
     through an input mechanism.
 </Requirement>
<Requirement id="2"> User history is tracked through an input mechanism, and it has to be
    retrieved afterward.
 </Requirement>
</Concern>

<Concern name="Persistence">
 <Requirement id="1">State-encoding is the conversion of application data to a format more
     suitable for storage in a database management system.
 </Requirement>
</Concern>

<Composition>
 <Requirement concern="Persistence" id="1">
  <Constraint action="ensure" operator="with">
   <Requirement concern="Context-awareness" id="2"/>
```

```
17   </Constraint>
18   <Outcome action="fulfilled"/>
19  </Requirement>
20 </Composition>
```

**Table 6**
Results of case study 1: MDSOCRE (M) versus MDSOCRE* (M-*), and statistical analysis of population independence of subjective evaluation.

| Interaction | Correctness | | Time | | Subjective evaluation | | |
|---|---|---|---|---|---|---|---|
| | M (%) | M-* (%) | M | M-* | Use M | Use M-* | p-value |
| Dependency | 85.71 | 71.42 | 3 m 43 s | 2 m 34 s | 2.28 | 4.14 | 0.006 |
| Reinforcement | 57.14 | 71.42 | 3 m | 3 m 09 s | 2.2 | 4 | 0.011 |
| Conflict | 57.14 | 85.71 | 3 m 51 s | 3 m | 1.8 | 4.14 | 0.002 |
| Mutex | 71.42 | 100 | 2 m 26 s | 1 m 43 s | 2 | 4.28 | 0.002 |
| Global | 64.28 | 85.71 | 13 m | 10 m 26 s | 2.10 | 4.14 | |

The overall organization of the experiment was as follows:

1. The four types of interactions were presented.
2. The MDSOCRE approach was explained, including usage examples.
3. The first set of tasks was given to the subjects for resolution.
4. After all the users finished solving the first set, the mappings for interactions, *i.e.,* MDSOCRE*, was explained.
5. A second set of tasks was given for resolution.

*Survey.* After all tasks were completed, study subjects were presented a questionnaire. It asked, for each interaction type, whether they would use a specific notation in the future to document this kind of interaction, graded on a five point Likert scale. Sentences followed the pattern below:

> I would use **MDSOCRE** to express *dependency—reinforcement—mutex—conflict* interactions in the future.
> I would use **MDSOCRE*** to express *dependency—reinforcement—mutex—conflict* interactions in the future.

*Results.* The results of the experiment regarding accuracy, time and subjective preference are presented in Table 6. Global results show that MDSOCRE* is amply more accurate (85.71% against 64.28%) for the whole experiment. Broken down by interaction type, MDSOCRE* is more accurate for *Reinforcement*, *Conflict* and *Mutex* interactions but slightly less accurate for *Dependency*. Regarding the time taken, MDSOCRE* provided significantly faster results, except for *Reinforcement*, where the time taken was slightly more than MDSOCRE. Overall time taken shows that MDSOCRE* is 24% faster than MDSOCRE. In the subjective preference score we see a strong bias toward MDSOCRE*, both per interaction type as globally. Moreover, statistical analysis reveals that the difference between the opinions for both notations is highly significant. We performed a population independence test[5] to establish this for each interaction type. The *p-values* for the tests show that with a 98% confidence level we can say that the opinion of the test subjects for MDSOCRE is different than for MDSOCRE*. Lastly, overall results show that the study subjects would use MDSOCRE* in the future, while they do not agree to use MDSOCRE in the future.

To conclude, the experiment shows that interpreting interactions in a requirement model written in MDSOCRE* is not only preferred by the users, but also significantly faster and also more accurate than when written in MDSOCRE.

### 5.2. Case study 2: MDSOCRE* versus MDSOCRE-i

The second experiment we performed evaluated the advantages of our extension to unmodified MDSOCRE. As the previous experiment showed that MDSOCRE* performs better than MDSOCRE, we chose to compare MDSOCRE* with MDSOCRE-i.

*Evaluators.* To avoid learning effects from the first experiment influencing the results of this experiment, a different group of test subjects was used. The group for this experiment consisted of 8 persons with industrial experience in the SM domain. All of them have worked for a company developing and testing SM software during an average of 4 years. They occupied different positions, *e.g.* testers, architects, developers. For this experiment we therefore used concerns and interactions taken from our SM requirement models written in MDSOCRE and MDSOCRE-i (included in Appendix B).

*Activities.* The experiment was organized as follows:

1. The four types of interactions were presented.
2. MDSOCRE approach was explained, including usage examples.
3. The mappings of MDSOCRE* for interactions were introduced.
4. MDSOCRE-i was introduced, including usage examples.
5. The two sets of tasks were delivered for their resolution.

[5] A two independent sample Wilcoxon rank sum test.

**Table 7**
Results of case study 2: MDSOCRE* (M-*) versus MDSOCRE-i (M-i), and analysis of population independence of subjective evaluation.

| Interaction | Correctness | | Time | | Subjective evaluation | | |
|---|---|---|---|---|---|---|---|
| | M-* (%) | M-i (%) | M-* | M-i | Use M* | Use M-i | p-value |
| Dependency | 75 | 87.5 | 4 m | 4 m 24 s | 2.5 | 3.5 | 0.36 |
| Reinforcement | 75 | 100 | 5 m 28 s | 2 m 23 s | 2.25 | 3.5 | 0.46 |
| Conflict | 87.5 | 100 | 2 m 22 s | 2 m 35 s | 2.5 | 3.87 | 0.007 |
| Mutex | 87.5 | 87.5 | 1 m 54 s | 1 m 59 s | 2.25 | 3.87 | 0.005 |
| Global | 81.25 | 93.75 | 13 m 44 s | 10 m 21 s | 2.37 | 3.68 | |

*Survey.* After the tasks were completed, a similar survey as in the first experiment was handed to the subjects (substituting MDSOCRE-i for MDSOCRE in the questions). This survey also included the following questions, allowing a response of either MDSOCRE* or MDSOCRE-i:

> Which notation makes interactions more evident?
> Which notation describes the interactions more clearly?

*Results.* The results for the second case study are presented in Table 7. Correctness is shown in the two first columns. Globally, MDSOCRE-i results in a higher accuracy than MDSOCRE*. Considering the interaction type, for *Dependency*, *Reinforcement* and *Conflict* MDSOCRE-i performed better while for *Mutex* the correctness results are the same. With respect to time, MDSOCRE-i was notably faster for *Reinforcement*, while for the other interactions the results are equivalent. In the overall time measure MDSOCRE-i was 32% faster than MDSOCRE*. Regarding subjective preference, users mildly agree to use MDSOCRE-i in the future while mildly disagreeing to use MDSOCRE* in the future. With less pronounced differences between the opinions of both notations, statistical analysis reveals that the difference between the opinions for both notations still is significant. The *p-values* for the tests show that with a 95% confidence level we can say that the opinion of the test subjects for MDSOCRE* is different than for MDSOCRE-i.

Finally, for the last two questions of the questionnaire: which tool makes interaction more evident and which tool describes interactions more clearly, on both questions MDSOCRE-i got 87.5% of preference compared to 12.5% obtained by MDSOCRE.

From the second experiment, we can conclude that MDSOCRE-i is preferred by the users, while being somewhat more accurate and significantly faster than MDSOCRE*.

Overall, considering the time and accuracy measurements of both experiments we can therefore conclude that for the interpretation of interactions in a requirements model MDSOCRE-i is significantly more accurate than MDSOCRE while also being much faster. Finally, from the subjective evaluation of MDSOCRE* being better than MDSOCRE, and of MDSOCRE-i being better than MDSOCRE*, we can infer that users largely prefer MDSOCRE-i over unmodified MDSOCRE (which includes MDSOCRE*).

*Time taken.* Using our extensions implies that more time needs to be invested in the explanation on how to use these new constructs. Even though we did not accurately measure the time needed for this *teaching* part of the experiment, we do have some observations in this regard. After the engineers were exposed to the limitations of the original approach and our workaround, they welcomed our extensions and quickly understood how to use them. Roughly speaking, the explanation of the extensions took one third of the time needed for the original introduction of the approach (which includes the workaround). Lastly, compared to the time to explain the extensions, the explanation of the workaround took more time, in addition to being more cumbersome and more error prone.

#### 5.2.1. Study validity: strong and weak points

*Strong point: industrial profile.* Because the MDSOCRE-i extension has been inspired by industrial experience in a complex domain, having the proposed extension validated by people in industry is fundamental. The strongest point of our study is therefore the industrial profile of the subjects.

In both cases the subjects had considerable experience in the industry. In the case of the first experiment, the test subjects belong to a company which develops mainly for the enterprise domain. They were accustomed to cope with interactions of requirements, frequently present in enterprise applications.

For the second case study, the subjects were experts for the particular domain of SMs. Moreover, they occupied different positions in the company, which reinforces the hypothesis about the ability of MDSOCRE-i to improve comprehension, not only for requirement engineers, but also for testers, developers and other profiles.

*Weak point: small scale.* The main weakness of both our studies is that they were performed with a small group of test subjects (7 and 8 respectively). Given the industrial setting we were unable to locate a larger group of people for both tests. Moreover for the second test the use of domain experts further restricted the set of possible test subjects.

However we do consider the study to be relevant because we see that the results are very consistent. Our assessment is confirmed at least regarding the user preference since the statistical analysis of the subjective evaluation returns highly

significant results. In other words, we can conclude that with a bigger sample size it is extremely likely that the user preferences would also be in favor of MDSOCRE-i.

In conclusion we state that despite the small number of subjects the results regarding time, accuracy and subjective evaluation demonstrate that MDSOCRE-i significantly outperforms MDSOCRE with respect to the comprehension of interactions in requirement models.

## 6. Related work

We have encountered two other comparative studies for AORE approaches but these however do not consider aspectual interactions. Sampaio et al. [27] analyze 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 of about 150 pages [15,16,21]. Chitchyan et. al. [10] 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 [7,25] 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 [33] 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. [9] moves the focus toward 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 remove (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 [3,18,32] 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.

Goal oriented requirement engineering, *e.g.* [22,31,34], deals with conflicts between goals (specially soft goals). Yu et al. [34], propose a systematic process to discover aspects from a goal model containing functional and non-functional goals. In this case conflicts arise during the iterative process of refining a goal graph, and they are solved by removing relationships that lead to a conflict. Related to this is the research on non-functional requirements (NFRs). NFRs cannot be fully satisfied, instead they can be "sufficiently" satisfied [12]. That is, non-functional goals can only be satisfied within acceptable limits [11]. The NFR framework establishes different types of contributions between goals. Goals and contributions are associated to labels indicating the degree of satisfaction or denial. These labels are propagated to determine the effect of different design decisions [13]. The engineer can then use this information to make design decisions reflecting the best trade-off for NFR.

In our domain, all of the requirements need to be fulfilled (none can be discarded), and their fulfillment is verified during a certification process carried out by specialized organizations (known as certification laboratories). Instead of removing requirements, we need to deal with conflicts by not allowing the activation or execution of conflicting features during a single system run. To illustrate this, consider the example in Section 2.3 regarding the conflicts originated by the activation of the Demo concern.

In relation to the ambiguity problem present in the SM domain it is worthwhile to mention the EA-Miner tool from Sampaio et al. [26]. This tool uses natural language processing techniques for partial automation of the identification of elements in the requirements, which leads to early aspect identification. This tool is intended to be of use for any AORE approach. This as it produces an output that can be consumed by other tools, for example ARCADE (which assists in the detection and resolution of conflicts) [25]. EA-Miner uses semantic tags to avoid the ambiguity problem. Nonetheless, it requires some interaction with the requirements engineer, especially in the presence of (noun-)phrases. In our case however, ambiguities come not only from the use of synonyms and (noun-)phrases, but also as full sentences that express similar ideas but in different forms.

## 7. Conclusions and future work

This paper presents our study of applicability of two AORE approaches: Theme/Doc [2] and MDSOCRE [20], 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 comprehensive support for our case, and proposed and validated extensions to both approaches to address this issue.

From our analysis we conclude that, considering both approaches without our extensions, MDSOCRE performs better than Theme/Doc. This because it allows to specify the composition of concerns in detail. We also 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 the analyst's domain knowledge. As we have different sources with different terminology, we found the MDSOCRE approach more suitable for our needs.

The first limitation of Theme/Doc is the lack of detailed information regarding the requirements participating in crosscutting relationships. To address this we added quantification labels that allow the analyst to provide more detail

to concern relationships. A second issue is the lack of support for expressing interactions between concerns. In order to solve this we introduced a new kind of interaction relationship that permits to express a conflict, mutex, dependency or reinforcement between two concerns, information that otherwise would be lost. This new interaction relationship can be combined with quantification labels to accurately document the requirements involved in the interaction.

For MDSOCRE, we were able to model interactions between two concerns using specific combinations of action and operator attributes in constraint specifications. As these combinations can be ambiguous and seemed not to be intuitive, we extended MDSOCRE with explicit support for interactions, called MDSOCRE-i. We then performed a user study to establish this ambiguity and to validate that MDSOCRE-i delivers faster results and aids in the understanding of interactions.

A first avenue of future work would be a study of the impact of our extensions to evolution of requirements. Second, the size of our requirements base and the fact that these come from different sources with their own formats, makes it desirable to automatize how the *modeled* version of the requirements is obtained and evolved from one version to another. We believe that extra information represented by the interactions can help to better track the impact of changes, but this needs to be corroborated.

Although we were able to improve the expressiveness of Theme/Doc, a detailed experiment could be useful to compare it with the results obtained in Section 4.1 for MDSOCRE-I. Also, the deployment of our extensions in the industry will help to analyze other issues such as scalability or requirement evolution.

A last path of future work consists of using the information regarding concern interactions – that now can be accurately recorded during the requirement engineering phase – in the design phase of the application. We expect that this information will aid in the development of the architectural and design artifacts that will allow the produced application to correctly address interactions between concerns.

## Appendix A. Theme/doc diagrams

See Figs. A.10 and A.11.

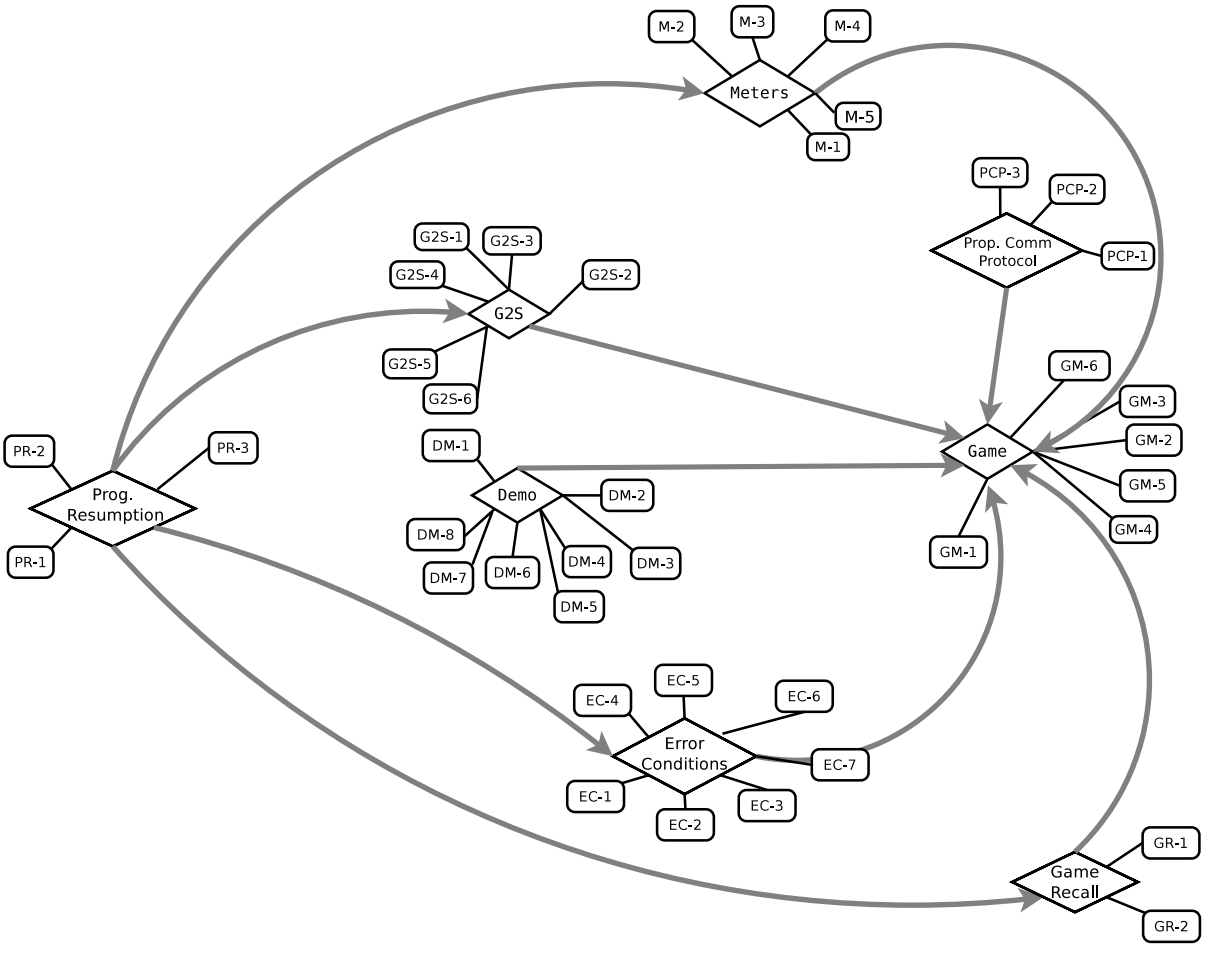


**Fig. A.10.** Theme/Doc approach applied to the selected requirements and concerns.

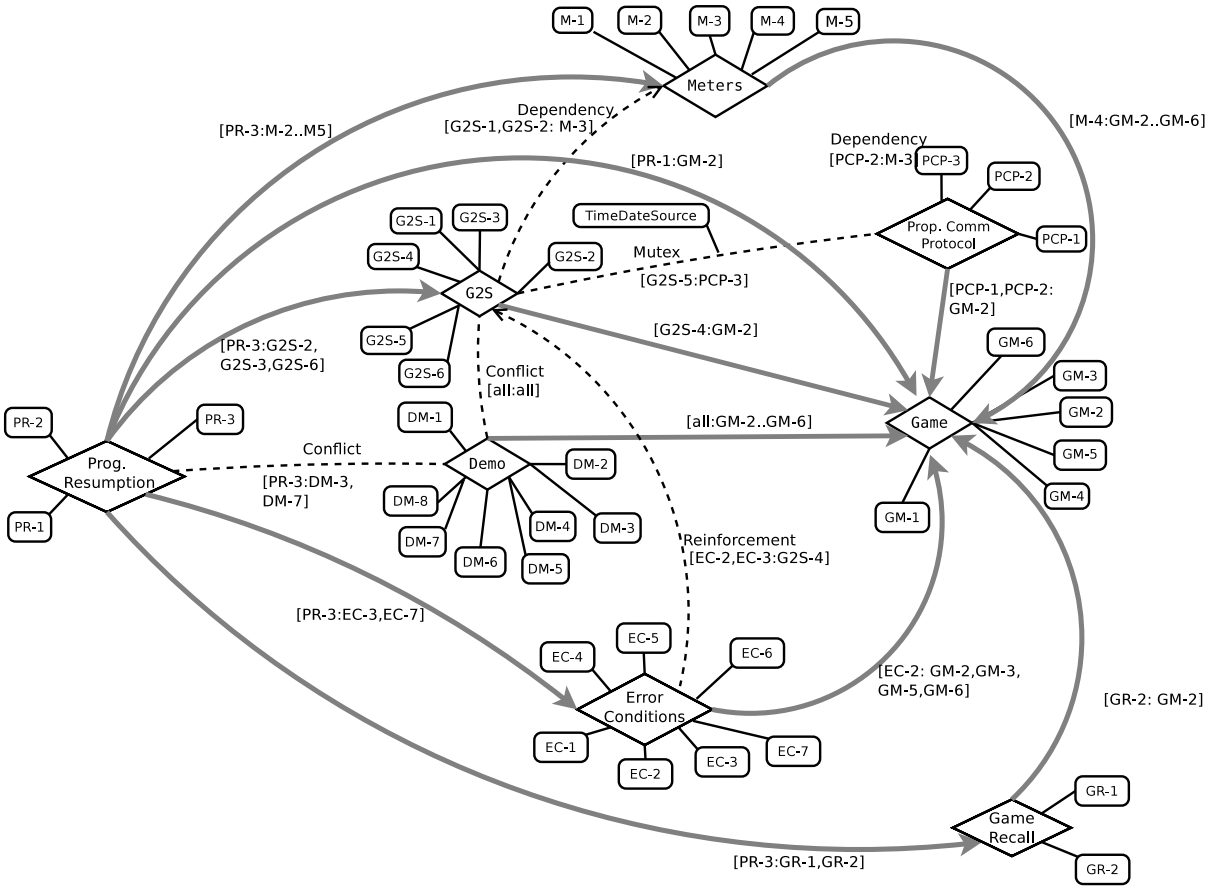


**Fig. A.11.** Extensions to Theme/Doc applied.

## Appendix B. MDSOCRE code listings

**Listing 7.** Concerns and crosscutting relationships for selected requirements

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <Concern name="Game">
        <Requirement id="1"> A slot machines have 5 reels.
        </Requirement>
        <Requirement id="2"> Reels spin when 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.
              It could be a ticket printer, a coin hopper.
        </Requirement>
    </Concern>

    <Concern name="Meters">
<!-- From GLI 11 -->
        <Requirement id="1"> Credit meter: shall at all times indicate all credits or cash
            available for the player to wager or cashout  (GLI 11 4.10.1)
        </Requirement>
        <Requirement id="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,bills, Ticket/Vouchers,
            coupons or other approved notes accepted. (GLI 11 4.10.5)
```

```
23          </Requirement>
24          <Requirement id="3"> Accounting Meters (GLI 11 4.10.9):  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.
25          </Requirement>
26          <Requirement id="4"> Meters should be updated upon occurrence of any event that must
                be counted, including: play, cashout, bill in, coin in.
27          </Requirement>
28  <!-- From G2S -->
29          <Requirement id="5" seeAlso="2,4"> Some 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
30          </Requirement>
31      </Concern>
32
33      <Concern name="Game Recall">
34          <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.
35          </Requirement>
36          <Requirement id="2"> Last play information shall provide all information 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.
37          </Requirement>
38      </Concern>
39
40      <Concern name="G2S">
41          <Requirement id="1"> The  G2S protocol is designed to communicate information between
                 a SM, and one or more host systems.
42          </Requirement>
43          <Requirement id="2"> Meter information can be queried by a host in real-time or a
                host may set a periodic meter subscription to cause the EGM to send selected
                meters at predetermined intervals.
44          </Requirement>
45          <Requirement id="3"> Information provided by the SM is used for audit purposes.
46          </Requirement>
47          <Requirement id="4"> The device can generate an event in an unsolicited manner or in
                response to a host command.
48          </Requirement>
49          <Requirement id="5"> Current timestamp can be configured by the host.
50          </Requirement>
51    <Requirement id="6"> Command GetGameRecallLog is used by a host to request the contents of
          a transaction log of last playsfrom a SM.
52    </Requirement>
53      </Concern>
54
55      <Concern name="Proprietary Communication Protocol">
56          <Requirement id="1"> The PCP  communicates a SM with a host system.
57          </Requirement>
58          <Requirement id="2"> It must report all meters of a SM.
59          </Requirement>
60          <Requirement id="3">Configuration settings such as current timestamp are configured
                from the host.
61          </Requirement>
62          <Requirement id="4"> If error conditions such as: door open, ticket inserted, paper
                out, etc. can be detected they shall be informed to the host.
63          </Requirement>
64      </Concern>
65
66      <Concern name="Program Resumption">
67      <!--From Nevada regulation -->
68      <Requirement id="1" seeAlso="3"> After a program interruption (e.g., processor reset),
            the software shall be able to recover to the state it was in immediately prior to the
             interruption occurring.
69           </Requirement>
70          <Requirement id="2"> Restoring Power. If a gaming device is powered down while in an
                error condition, then upon restoring power, the specific error message shall
                still be displayed and the gaming device shall remain locked-up.
71          </Requirement>
72       <!-- From G2S  1.16 -->
```

```
73          <Requirement id="3" seeAlso="1"> A SM must store all meter information in persistent
                memory.
74          </Requirement>
75      </Concern>
76
77      <Concern name="Error Conditions and Events">
78       <!--From Nevada regulation -->
79          <Requirement id="1"> Gaming devices shall be capable of detecting and displaying
                error conditions and illuminate the tower light for each  or sound an audible
                alarm.
80          </Requirement>
81          <Requirement id="2"> Error conditions should cause the gaming device to lock up and
                require attendant intervention. Error conditions shall be cleared either by an
                attendant or upon initiation of a new play sequence after the error has cleared
                except for those deemed as a critical error.
82          </Requirement>
83         <!-- From GLI  -->
84          <Requirement id="3"> Error conditions are: coin jam, reverse coin in, stacker full,
                bill jam, external doors open.
85          </Requirement>
86          <Requirement id="4"> Video based games shall display meaningful text as to the error
                 conditions.
87          </Requirement>
88          <Requirement id="5"> Error conditions shall be communicated to an on-line monitoring
                and  control system when this is available.
89          </Requirement>
90          <!-- From g2s -->
91          <Requirement id="6" seeAlso="1">  An event represents an occurrence of an incident
                detected by a device in an EGM
92          </Requirement>
93          <Requirement id="7"> Important events must be reported in real-time, including: error
                 conditions, tickets inserted, ticket printed.
94          </Requirement>
95      </Concern>
96
97      <Concern name="Demo Mode">
98        <!-- From Nevada -->
99          <Requirement id="1"> The Slot Machine must permit a a test, diagnostic or demo mode,
                which permits gaming device (e.g.,  a hopper test) shall be completed on
                resumption of normal operation.
100         </Requirement>
101         <Requirement id="2"> If the gaming device is in a test, diagnostic or demo mode, any
                test that incorporates credits entering or leaving the gaming device (e.g.,  a
                hopper test) shall be completed on resumption of normal operation.
102         </Requirement>
103         <Requirement id="3"> There shall not be any mode other than normal operation (ready
                for play) that increments any of the electronic meters.
104         </Requirement>
105         <Requirement id="4">  Any credits on the gaming device that were added during the
                test, diagnostic or demo mode shall be automatically cleared before the mode is
                exited. </Requirement>
106         <!-- From GLI 11 4.17 -->
107         <Requirement id="5"> Specific meters are permissible for these types of modes
                provided the meters indicate as such
108         </Requirement>
109         <Requirement id="6"> The main cabinet door of the gaming device may automatically
                place the gaming device in a service or test/diagnostic mode. Test/diagnostics
                mode may also be entered, via an appropriate instruction, from an attendant
                during an audit mode access. These modes should not be accessible to the player
110         </Requirement>
111         <Requirement id="7">  When exiting from test-diagnostic mode, the game shall return
                to the original state it was in when the test mode was entered
112         </Requirement>
113         <Requirement id="8"> Test Games. If the device is in a game test mode, the machine
                shall clearly indicate that it is in a test mode, not normal play.
114         </Requirement>
115     </Concern>
116
117 <!--        CROSSCUTTING RELATIONSHIPS      -->
118
119     <Composition>
120         <Requirement concern="Demo Mode" id="all">
121             <Constraint action="enforce" operator="on">
122                 <Requirement concern="Game" id="all" />
```

```
123             </Constraint>
124             <Constraint action="exclude" operator="on">
125                 <Requirement concern="Game" id="1" />
126                 <Requirement concern="Game" id="2" />
127             </Constraint>
128             <Outcome action="fulfilled"/>
129         </Requirement>
130     </Composition>
131
132
133     <Composition>
134         <Requirement concern="Game Recall" id="2">
135             <Constraint action="enforce" operator="on">
136                 <Requirement concern="Game" id="2" />
137             </Constraint>
138             <Outcome action="fulfilled"/>
139         </Requirement>
140     </Composition>
141
142
143     <Composition>
144         <Requirement concern="Meters" id="4">
145             <Constraint action="enforce" operator="on">
146                 <Requirement concern="Game" id="2,3,4,5,6" />
147             </Constraint>
148             <Outcome action="fulfilled"/>
149         </Requirement>
150
151     </Composition>
152
153
154     <Composition>
155         <Requirement concern="Program Resumption" id="1">
156             <Constraint action="enforce" operator="on">
157                 <Requirement concern="Game" id="2" />
158             </Constraint>
159             <Outcome action="fulfilled"/>
160         </Requirement>
161     </Composition>
162
163
164     <Composition>
165         <Requirement concern="G2S" id="4">
166             <Constraint action="enforce" operator="on">
167                 <Requirement concern="Game" id="2" />
168             </Constraint>
169             <Outcome action="fulfilled"/>
170         </Requirement>
171     </Composition>
172
173     <Composition>
174         <Requirement concern="Error Condition " id="3">
175             <Constraint action="enforce" operator="on">
176                 <Requirement concern="Game" id="2,3,5,6" />
177             </Constraint>
178             <Outcome action="satisfied">
179                 <Requirement concern=" Error Condition" id="2"/>
180                 </Outcome>
181             </Requirement>
182         </Composition>
183
184         <Composition>
185             <Requirement concern="Proprietary Communication Protocol" id="1,2">
186                 <Constraint action="enforce" operator="on">
187                     <Requirement concern="Game" id="2" />
188                 </Constraint>
189                 <Outcome action="fulfilled"/>
190             </Requirement>
191         </Composition>
192
193 </root>
```

**Listing 8**. Interactions for the selected requirements expressed using MDSOCRE notation

```
<Composition>
 <Requirement concern="Meters" id="3">
  <Constraint action="ensure" operator="with">
   <Requirement concern="G2S" id="1,2"/>
  </Constraint>
  <Outcome action="fulfilled"/>
 </Requirement>
</Composition>
<Composition>
 <Requirement concern="Meters" id="3">
  <Constraint action="ensure" operator="with">
   <Requirement concern="PCP" id="1,2"/>
  </Constraint>
  <Outcome action="fulfilled"/>
 </Requirement>
</Composition>
<Composition>
 <Requirement concern="G2S" id="7">
   <Constraint action="enforce" operator="xor">
   <Requirement concern="Proprietary Communication Protocol" id="5"/>
  </Constraint>
  <Outcome action="fulfilled"/>
 </Requirement>
</Composition>
<Composition>
 <Requirement concern="Error Condition" id="2,3">
  <Constraint action="provide" operator="for">
   <Requirement concern="G2S" id="4"/>
  </Constraint>
  <Outcome action="fulfilled"/>
 </Requirement>
</Composition>
<Composition>
 <Requirement concern="G2S" id="3">
   <Constraint action="enforce" operator="xor">
   <Requirement concern="Proprietary Communication Protocol" id="4"/>
  </Constraint>
  <Outcome action="fulfilled"/>
 </Requirement>
</Composition>
<Composition>
 <Requirement concern="GameRecall" id="1,2">
  <Constraint action="ensure" operator="with">
   <Requirement concern="G2S" id="6"/>
  </Constraint>
  <Outcome action="fulfilled"/>
 </Requirement>
</Composition>
<Composition>
 <Requirement concern="PCP" id="all">
   <Constraint action="enforce" operator="xor">
   <Requirement concern="Demo" id="all"/>
  </Constraint>
  <Outcome action="fulfilled"/>
 </Requirement>
</Composition>
<Composition>
 <Requirement concern="G2S" id="all">
   <Constraint action="enforce" operator="xor">
   <Requirement concern="Demo" id="all"/>
  </Constraint>
  <Outcome action="fulfilled"/>
 </Requirement>
</Composition>
```

**Listing 9**. Interactions for the selected requirements expressed using MDSOCRE-i notation

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

```
                    <Requirement concern="Meters" id="3"/>
                </Interaction>
            </Requirement>
        </Composition>
        <Composition>
            <Requirement concern="Proprietary Communication Protocol" id="1,2">
                <Interaction type="dependency">
                    <Requirement concern="Meters" id="3"/>
                </Interaction>
            </Requirement>
        </Composition>
        <Composition>
            <Requirement concern="Proprietary Communication Protocol" id="5">
                <Interaction type="mutex">
                    <Requirement concern="G2S" id="7" />
                </Interaction>
            </Requirement>
        </Composition>
        <Composition>
            <Requirement concern="Error Condition" id="2,3">
                <Interaction type="reinforcement" >
                    <Requirement concern="G2S" id="4"/>
                </Interaction>
            </Requirement>
        </Composition>
        <Composition>
            <Requirement concern="Proprietary Communication Protocol" id="4">
                <Interaction type="mutex" >
                    <Requirement concern="G2S" id="3"/>
                </Interaction>
            </Requirement>
        </Composition>
  <Composition>
    <Requirement concern="G2S" id="6">
      <Interaction type="dependency">
       <Requirement concern="GameRecall" id="1,2"/>
      </Interaction>
    </Requirement>
        </Composition>
        <Composition>
            <Requirement concern="Proprietary Communication Protocol" id="all">
                <Interaction type="conflict" >
                    <Requirement concern="Demo" id="all"/>
                </Interaction>
            </Requirement>
        </Composition>
        <Composition>
            <Requirement concern="G2S" id="all">
                <Interaction type="conflict" >
                    <Requirement concern="Demo" id="all"/>
                </Interaction>
            </Requirement>
        </Composition>
```

## References


[1] Recommended practice for software requirements specifications, IEEE Std 830-1998, 1998.
[2] Elisa Baniassad, Siobhan Clarke, Finding aspects in requirements with Theme/Doc, in: Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design, Workshop of the 3rd International Conference on Aspect-Oriented Software Development, March 2004.
[3] Elisa Baniassad, Siobhan Clarke, Theme: an approach for aspect-oriented analysis and design, in: ICSE'04: Proceedings of the 26th International Conference on Software Engineering, Washington, DC, USA, IEEE Computer Society, 2004, pp. 158–167.
[4] Elisa Baniassad, Paul C. Clements, Joao Araujo, Ana Moreira, Awais Rashid, Bedir Tekinerdogan, Discovering early aspects, IEEE Softw. 23 (1) (2006) 61–70.
[5] David Bar-On, Shmuel Tyszberowicz, Derived requirements generation: the DRAS methodology, in: Software Science, Technology and Engineering, IEEE International Conference on, 0, 2007, pp. 116–126.
[6] Isabel Brito, Ana 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] Isabel Sofia Brito, Filipe Vieira, Ana Moreira, Rita Almeida Ribeiro, Handling conflicts in aspectual requirements compositions, Transactions in Aspect-Oriented Software Development 3 (2007) 144–166.
[8] Ruzanna Chitchyan, Johan Fabry, Shmuel Katz, Arend Rensink, Editorial for special section on dependencies and interactions with aspects 5490 (2009) 133–134.
[9] Ruzanna Chitchyan, Awais Rashid, Paul Rayson, Robert Waters, Semantics-based composition for aspect-oriented requirements engineering, in: AOSD'07: Proceedings of the 6th International Conference on Aspect-Oriented Software Development, New York, NY, USA, ACM, 2007, pp. 36–48.

[10] Ruzanna Chitchyan, Awais Rashid, Peter Sawyer, Comparing requirement engineering approaches for handling crosscutting concerns, in: João Araújo, Amador Durán Toro, João Falcão e Cunha (Eds.), 8th Workshop on Requirements Engineering held at CAiSE'05, 2005, pp. 1-12..
[11] Lawrence Chung, Brian A. Nixon, Dealing with non-functional requirements: three experimental studies of a process-oriented approach, in: Proceedings of the 17th International Conference on Software Engineering, ICSE'95, New York, NY, USA, ACM, 1995, pp. 25–37.
[12] Lawrence Chung, Brian A. Nixon, Eric Yu, John Mylopoulos, Non-Functional Requirements in Software Engineering, in: International Series in Software Engineering, vol. 5, Springer, 1999.
[13] Lawrence Chung, Julio Cesar Prado Leite, Conceptual modeling: Foundations and applications, in: Non-Functional Requirements in Software Engineering, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 363–379.
[14] Siobhán Clarke, Elisa Baniassad, Aspect-Oriented Analysis and Design. The Theme Approach, in: Object Technology Series, Addison-Wesley, Boston, USA, 2005.
[15] Gaming Laboratories International, Gaming Devices in Casinos, 2007. Available at: http://www.gaminglabs.com/.
[16] Gaming Standard Association, Game to Server (G2S) Protocol Specification, 2008. Available at: http://www.gamingstandards.com/.
[17] John C. Grundy, Aspect-oriented requirements engineering for component-based software systems, in: RE'99: Proceedings of the 4th IEEE International Symposium on Requirements Engineering, Washington, DC, USA, IEEE Computer Society, 1999, pp. 84–91.
[18] Ivar Jacobson, Pan-Wei Ng, Aspect-Oriented Software Development with Use Cases, in: Addison-Wesley Object Technology Series, Addison-Wesley Professional, 2004.
[19] Jia Liu, Don S. Batory, Srinivas Nedunuri, Modeling interactions in feature oriented software designs, in: Feature Interactions in Telecommunications and Software Systems VIII, 2005, pp. 178–197.
[20] A. Moreira, A. Rashid, J. Araujo, Multi-dimensional separation of concerns in requirements engineering, in: Proc. 13th IEEE International Conference on Requirements Engineering, 29 Aug.–2 Sept. 2005, pp. 285–296.
[21] Nevada Gaming Commission, Technical Standards For Gaming Devices and On-Line Slot Systems, 2008. Available at: http://gaming.nv.gov/stats_regs.htm.
[22] Nan Niu, Yijun Yu, Bruno González-Baixauli, Neil A. Ernst, Julio Cesar Sampaio, do Prado Leite, John Mylopoulos, Aspects across software life cycle: a goal-driven approach, T. Aspect-Oriented Software Development VI 6 (2009) 83–110.
[23] Balasubramaniam Ramesh, Matthias Jarke, Toward reference models for requirements traceability, IEEE Trans. Softw. Eng. 27 (2001) 58–93.
[24] Awais Rashid, Ana Moreira, Domain models are NOT aspect free, in: ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, MODELS06, in: Lecture Notes in Computer Science, vol. 4199, Springer Verlag, 2006, pp. 155–169.
[25] Awais Rashid, Ana Moreira, João Araújo, Modularisation and composition of aspectual requirements, in: Proceedings of the 2nd International Conference on Aspect-oriented Software Development, AOSD'03, New York, NY, USA, ACM, 2003, pp. 11–20.
[26] Americo Sampaio, Awais Rashid, Ruzanna Chitchyan, Paul Rayson, Ea-miner: towards automation in aspect-oriented requirements engineering, in: Awais Rashid, Mehmet Aksit (Eds.), Transactions on Aspect-Oriented Software Development III, in: Lecture Notes in Computer Science, vol. 4620, Springer, Berlin, Heidelberg, 2007, pp. 4–39.
[27] Americo Sampaio, Phil Greenwood, Alessandro F. Garcia, Awais 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, Washington, DC, USA, IEEE Computer Society, 2007, pp. 166–175.
[28] Frans Sanen, Eddy Truyen, Bart De Win, Wouter Joosen, Neil Loughran, Geoff Coulson, Awais Rashid, Andronikos Nedos, Andrew Jackson, Siobhan Clarke, Study on interaction issues, Technical Report AOSD-Europe Deliverable D44, AOSD-Europe-KUL-7, Katholieke Universiteit Leuven, 28 February 2006.
[29] Ian Sommerville, Ian Sommerville, Pete Sawyer, Pete Sawyer, Viewpoints: principles, problems and a practical approach to requirements engineering, Annals of Software Engineering 3 (1997) 101–130.
[30] Peri Tarr, Harold Ossher, William Harrison 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, Los Alamitos, CA, USA, IEEE Computer Society Press, 1999, pp. 107–119.
[31] Axel Van Lamsweerde, Goal-oriented requirements engineering: A guided tour, in: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering, RE'01, Washington, DC, USA, IEEE Computer Society, 2001, p. 249.
[32] Jon Whittle, João Araújo, Scenario modelling with aspects, IEE Proceedings - Software 151 (4) (2004) 157–172.
[33] Jon Whittle, Praveen Jayaraman, MATA: A tool for aspect-oriented modeling based on graph transformation, in: Holger Giese (Ed.), Models in Software Engineering: Workshops and Symposia at MoDELS 2007, in: Lecture Notes in Computer Science, vol. 5002, Springer, Berlin, Heidelberg, 2008, pp. 16–27.
[34] Yijun Yu, Julio Cesar, Sampaio do Prado Leite, John Mylopoulos, From goals to aspects: discovering aspects from requirements goal models, in: Proceedings of the Requirements Engineering Conference, 12th IEEE International, Washington, DC, USA, IEEE Computer Society, 2004, pp. 38–47.
[35] Arturo Zambrano, Johan Fabry, Guillermo Jacobson, 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, ACM Press, 2010, pp. 2161–2168.