

# Modularisation and Composition of Aspectual Requirements

Awais Rashid  
Computing Department  
Lancaster University  
Lancaster LA1 4YR, UK  
+44-1524-592647

awais@comp.lancs.ac.uk

Ana Moreira  
Dept. Informática  
FCT, Universidade Nova de Lisboa  
2829-516 Caparica, Portugal  
+351-21-2948536

amm@di.fct.unl.pt

João Araújo  
Dept. Informática  
FCT, Universidade Nova de Lisboa  
2829-516 Caparica, Portugal  
+351-21-2948536

ja@di.fct.unl.pt

## ABSTRACT

An effective requirements engineering (RE) approach must harmonise the need to achieve separation of concerns with the need to satisfy broadly scoped requirements and constraints. Techniques such as use cases and viewpoints help achieve separation of stakeholders' concerns but ensuring their consistency with global requirements and constraints is largely unsupported. In this paper we propose an approach to modularise and compose such crosscutting, aspectual requirements. The approach is based on separating the specification of aspectual requirements, non-aspectual requirements and composition rules in modules representing coherent abstractions and following well-defined templates. The composition rules employ informal, and often concern-specific, actions and operators to specify how an aspectual requirement influences or constrains the behaviour of a set of non-aspectual requirements. We argue that such modularisation makes it possible to establish early trade-offs between aspectual requirements hence providing support for negotiation and subsequent decision-making among stakeholders. At the same time early separation of crosscutting requirements facilitates determination of their mapping and influence on artefacts at later development stages. A realisation of the proposed approach, based on viewpoints and the eXtensible Markup Language (XML), supported by a tool called ARCaDe and a case study of a toll collection system is presented.

## Keywords

Aspect-oriented requirements engineering, aspect composition, aspectual trade-offs, traceability

## 1. INTRODUCTION

Aspect-oriented software development (AOSD) [12] aims at addressing crosscutting concerns by providing means for their systematic identification, separation, representation and composition. Crosscutting concerns are encapsulated in separate

modules, known as aspects, so that localisation can be promoted. This results in better support for modularisation hence reducing development, maintenance and evolution costs. A number of aspect-oriented programming (AOP) approaches have been proposed. These range from language mechanisms [1] to filter-based techniques [5] through to traversal-oriented [20] and multi-dimensional approaches [23, 28]. Work has also been carried out to incorporate aspects, and hence separation of crosscutting concerns, at the design level mainly through extensions to the UML meta-model e.g. [8, 9, 27]. Research on the use of aspects at the requirements engineering stage is still immature and there is no consensus about what an aspect is at this early stage of software development and how it maps to artefacts at later development stages.

The focus of this paper is on modularisation and composition of requirements level concerns that cut across other requirements. These crosscutting concerns are responsible for producing tangled representations that are difficult to understand and maintain. Examples of such concerns at the requirements level are compatibility, availability and security requirements that cannot be encapsulated by a single viewpoint [13] or use case [17] and are typically spread across several of them. There is, therefore, a need to include aspects as fundamental modelling primitives at the requirements engineering level. The motivations for this are two-fold:

1. Providing improved support for separation of crosscutting functional and non-functional properties during requirements engineering hence offering a better means to identify and manage conflicts arising due to tangled representations;
2. Identifying the mapping and influence of requirements level aspects on artefacts at later development stages hence establishing critical trade-offs before the architecture is derived.

This paper proposes an approach aimed as a stepping-stone towards the above goals. Section 2 provides some background on existing approaches to manage crosscutting concerns at the requirements level. Section 3 presents a general model that supports effective identification and specification of aspectual requirements and their mapping and influence on later development stages. Section 4 instantiates the general model with a concrete set of techniques, namely viewpoints and XML. The concrete approach is supported by the Aspectual Requirements Composition and Decision support tool (ARCaDe) and described through its application to a case study of a toll collection system.

Section 5 introduces some related work, and finally, section 6 concludes the paper by discussing key outstanding issues and directions for future work.

## 2. BACKGROUND

In RE viewpoints [13], use cases [17] and goals [19] have been advocated as a means of partitioning requirements as a set of partial specifications that aid traceability and consistency management. However, ensuring the consistency of these partial specifications with global requirements and constraints is largely unsupported.

An aspect-oriented requirements engineering approach targeted to component based software development has been proposed in [15]. There is a characterisation of diverse aspects of a system that each component provides to end users or other components. However, the identification of aspects for each component is not clearly defined.

The XBel framework [4] offers a logic-based approach, supported by a model checker, for merging and reasoning about multiple, inconsistent viewpoints. Although the framework can be used to support requirements negotiation, the focus is on reasoning about the properties of the specification in the presence of inconsistent viewpoints and not on providing means for explicit modularisation and composition of crosscutting requirements. Nor is there any support for identifying the mapping and influence of crosscutting requirements on later development stages.

Separation of crosscutting properties has also been considered in [26] which proposes a viewpoint-oriented requirements engineering method called PREView. A PREView viewpoint encapsulates partial information about the system. Requirements are organised in terms of several viewpoints, and analysis is conducted against a set of *concerns*<sup>1</sup> intended to correspond broadly to the overall system goals. Due to this broad scope *concerns* crosscut the requirements emerging from viewpoints. In applications of the method, the *concerns* that are identified are typically high-level non-functional requirements. Beyond alerting the requirements engineer to the risk that viewpoint requirements and *concerns* may cause inconsistencies, the approach does not identify the mapping or influence of crosscutting properties on artefacts at later development stages.

## 3. GENERIC MODEL FOR AORE

Modern systems have to run in highly volatile environments where the business rules change rapidly. Therefore, systems must be easy to adapt and evolve. If not handled properly, crosscutting concerns inhibit adaptability. It is therefore essential to think about crosscutting concerns as early as possible. The model we envisage to deal with crosscutting concerns at the requirements level is illustrated in figure 1. It is a refinement of the Aspect-Oriented Requirements Engineering (AORE) model we presented in [24] which is based on treating PREView *concerns* as adaptations of the AOP notion of aspects. However, the model in [24] maintains composition relationships within the aspect definitions themselves and that too at a coarse granularity as in PREView. Consequently, it is only possible to see how an aspect

affects a set of viewpoints. The influence of an aspectual requirement and the constraints it imposes on specific requirements within the viewpoints affected by the aspect cannot be determined. Furthermore, the model in [24] does not incorporate explicit support for negotiating requirements based on aspectual trade-offs and revising the specification.

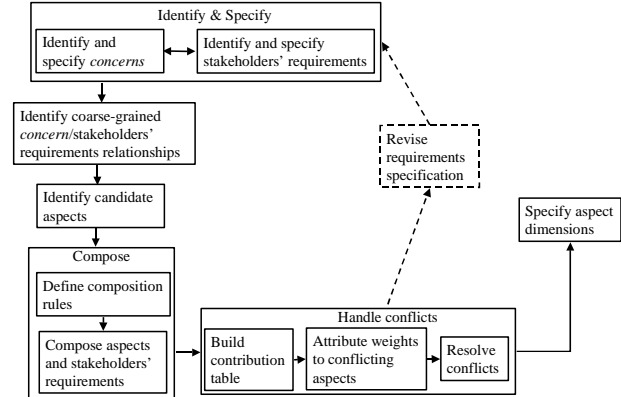


Figure 1: AORE model

We start by identifying and specifying both *concerns* and stakeholders' requirements. The latter is carried out using an existing requirements level separation of concerns mechanism such as viewpoints [13], use cases [17], goals [19] or problem frames [16]. The order in which the specification of *concerns* and stakeholders' requirements is accomplished is dependant on the dynamics of the interaction between requirements engineers and the stakeholders. In any case, it is useful to relate *concerns* to requirements, through a matrix, as the former may constrain the latter. Looking at the matrix (cf. table 1) we can see which *concerns* crosscut the modules encapsulating stakeholders' requirements (denoted by SR) and qualify as *candidate aspects* (we explain the rationale behind the notion of *candidate aspects* later in this section).

Table 1. Relating *concerns* with stakeholder requirements

SR \ Concerns	SR <sub>1</sub>	SR <sub>2</sub>	...	SR <sub>n</sub>
Concern <sub>1</sub>				✓
Concern <sub>2</sub>		✓		✓
...				
Concern <sub>n</sub>	✓	✓		

Once the coarse-grained relationships between *concerns* and stakeholders' requirements have been established and the candidate aspects identified, the next step is to define detailed composition rules. These rules operate at the granularity of individual requirements and not just the modules encapsulating them. Consequently, it is possible to specify how an aspectual requirement influences or constrains the behaviour of a set of non-aspectual requirements in various modules. At the same time, if desired, aspectual trade-offs can be observed at a finer granularity. This alleviates the need for unnecessary negotiations among stakeholders for cases where there might be an apparent trade-off between two (or more) aspects but in fact different, isolated requirements are being influenced by them. It also facilitates identification of individual, conflicting aspectual requirements

<sup>1</sup> From this point onwards the term *concerns* in italics is used to refer to the PREView notion of concerns and not concerns in software engineering in general.

with respect to which negotiations must be carried out and trade-offs established.

After composing the candidate aspects and stakeholders' requirements using the composition rules, identification and resolution of conflicts among the candidate aspects is carried out. This is accomplished by:

- (i) Building a contribution matrix (cf. table 2) where each aspect may contribute negatively (-) or positively (+) to the others (empty cells represent "don't care" contributions).
- (ii) Attributing weights to those aspects that contribute negatively to each other in relation to a set of stakeholders' requirements. Each weight is a real number in the interval [0 .. 1] and represents the priority of an aspect in relation to a set of stakeholders' requirements.
- (iii) Solving the conflicts with the stakeholders, using the above prioritisation approach to help communication.

**Table 2. Contributions between candidate aspects**

Aspects \ Aspects	Aspect <sub>1</sub>	Aspect <sub>2</sub>	...	Aspect <sub>n</sub>
Aspect <sub>1</sub>		+		
Aspect <sub>2</sub>				-
...				
Aspect <sub>n</sub>				

Conflict resolution might lead to a revision of the requirements specification (stakeholders' requirements, aspectual requirements or composition rules). If this happens, then the requirements are recomposed and any further conflicts arising are resolved. The cycle is repeated until all conflicts have been resolved through effective negotiations.

The last activity in the model is identification of the dimensions of an aspect. We have observed that aspects at this early stage can have an impact that can be described in terms of two dimensions [24]:

- *Mapping*: an aspect might map onto a system feature/function (e.g. a simple method, object or component), decision (e.g. a decision for architecture choice) and design (and hence implementation) aspect (e.g. response time). This is the reason we have chosen to call aspects at the RE stage *candidate aspects* as, despite their crosscutting nature at this stage, they might not directly map onto an aspect at later stages.
- *Influence*: an aspect might influence different points in a development cycle, e.g. availability influences the system architecture while response time influences both architecture and detailed design.

#### 4. CONCRETE MODEL WITH VIEWPOINTS AND XML

The concrete techniques we have chosen are viewpoints [13] for identifying the stakeholder requirements, and XML as the definition language to specify these requirements, the candidate aspects identified and the composition rules to relate viewpoints with aspects. Tool support is provided by the Aspectual Requirements Composition and Decision support tool, ARCaDe. The tool makes it possible to define the viewpoint requirements, aspectual requirements and composition rules using pre-defined templates. These templates can, optionally, be enforced using

XML schemas. The modules encapsulating the various requirements and composition rules are stored in eXist, a native XML database system [3]. A combination of DOM (Document Object Model) and SAX (Simple API for XML) is employed to:

- validate the composition rules i.e. to ensure that they refer to viewpoints, aspects and requirements that exist in the database;
- compose the aspects and viewpoints and identify resulting conflicts in order to establish trade-offs.

Our choice of viewpoints as a mechanism to specify stakeholders' requirements is driven by our previous experience in handling global requirements in viewpoint-oriented requirements engineering [24]. Instead of viewpoints we could have used other requirements approaches such as: goal-oriented requirements which cover functional and non-functional *concerns* [19]; use cases or scenario-based approaches, by specifying which use cases/scenarios are crosscut by a *concern* [25]; problem frames, which can be viewed as *concerns* [16]. XML has been chosen because, as demonstrated by the following case study, there is a need for concern-specific actions and composition operators when defining the composition rules. The extensible model offered by XML coupled with the rich specification model of the XML schema language makes it an ideal choice as it is virtually impossible to anticipate the various types of composition operators and actions that might be required. Since the XML schema language is extensible – it is based on XML itself – it is possible to enforce constraints on the specification of composition rules when new operators and/or actions are introduced. Furthermore, the ability to define semantically meaningful tags and informal operators ensures that the readability of the requirements specification is not compromised as the specification resides in the stakeholders' domain and must be readable by them. The following case study illustrates this concrete realisation of the generic AORE model.

#### 4.1 Case Study

The case study we have chosen is a simplified version of the toll collection system on the Portuguese highways [6]:

"In a road traffic pricing system, drivers of authorised vehicles are charged at toll gates automatically. The gates are placed at special lanes called green lanes. A driver has to install a device (a gizmo) in his/her vehicle. The registration of authorised vehicles includes the owner's personal data, bank account number and vehicle details. The gizmo is sent to the client to be activated using an ATM<sup>2</sup> that informs the system upon gizmo activation.

A gizmo is read by the toll gate sensors. The information read is stored by the system and used to debit the respective account.

When an authorised vehicle passes through a green lane, a green light is turned on, and the amount being debited is displayed. If an unauthorised vehicle passes through it, a yellow light is turned on and a camera takes a photo of the plate (used to fine the owner of the vehicle). There are three types of toll gates: single toll, where the same type of vehicles pay a fixed amount, entry toll to enter a motorway and exit toll to leave it. The amount paid on motorways depends on the type of the vehicle and the distance travelled."

<sup>2</sup> Portuguese ATMs offer a wide range of services, e.g. selling train or theatre tickets.

### 4.1.1 Identify and specify stakeholders' requirements

The following viewpoints can be identified. Note that some of the viewpoints have sub-viewpoints:

- *ATM*: allows customers to enter their own transactions using cards. The ATM sends the transaction information for validation and processing.
- *Vehicle*: enters and leaves toll gates. There is a sub-viewpoint *Unauthorised Vehicle* whose plate number is photographed.
- *Gizmo*: is read by the system and is glued on the windscreen of the car it belongs to.
- *Police*: receives information about the unauthorised vehicles and their infractions.
- *Debiting System*: interacts with the bank to allow the system to debit client accounts.
- *Toll Gate*: through which the vehicles pass when entering or leaving the toll collection system. There are two sub-viewpoints: *Entry Toll* and *Paying Toll*. *Entry Toll* detects gizmos. The *Paying Toll* viewpoint is further refined into two sub-viewpoints: *Single Toll* turns the light green and displays the amount to be paid for authorised vehicles and turns the light yellow, sounds an alarm and photographs the plate numbers for unauthorised vehicles; *Exit Toll* behaves similarly to single toll, except that it must take into account the valid (or invalid) entrance of the vehicle.
- *Vehicle Owner*: who has three sub-viewpoints: *Registration* of vehicles, cancellation of registration and modification of registration details; *Billing* in the form of regular invoices and *Activation* of the gizmo using ATMs.
- *System administrator*: introduces new information and modifies existing information in the system.

Throughout the paper we will use the viewpoints *ATM*, *Vehicle*, *Gizmo* and *Toll Gate* to illustrate our approach. Figures 2 through 5 show these viewpoints in XML. The structure is self-explanatory. A *Viewpoint* tag denotes the start of a viewpoint while a *Requirement* tag denotes the start of a requirement. Refinements such as sub-viewpoints and sub-requirements are represented via the nesting of the tags. Each requirement has an *id* which is unique within its defining scope i.e. the viewpoint. Viewpoint names are unique within each case study in ARCADE. However, XML namespaces can be used for the purpose as well.

```
<?xml version="1.0" ?>
<Viewpoint name="ATM">
  <Requirement id="1">
    The ATM sends the customer's card number, account number and gizmo
    identifier to the system for activation and reactivation.
    <Requirement id="1.1">
      The ATM is notified if the activation or reactivation was successful or
      not.
      <Requirement id="1.1.1">In case of unsuccessful activation or
      reactivation the ATM is notified of the reasons for
      failure.</Requirement>
    </Requirement>
  </Requirement>
</Viewpoint>
```

Figure 2: The ATM viewpoint in XML

```
<?xml version="1.0" ?>
<Viewpoint name="Vehicle">
  <Requirement id="1">The vehicle enters the system when it is within ten
  meters of the toll gate.</Requirement>
```

```
<Requirement id="2">The vehicle enters the toll gate.</Requirement>
<Requirement id="3">The vehicle leaves the toll gate.</Requirement>
<Requirement id="4">The vehicle leaves the system when it is twenty
meters away from the toll gate.</Requirement>
<Viewpoint name="UnauthorisedVehicle">
  <Requirement id="1">The vehicle number plate will be
  photographed.</Requirement>
</Viewpoint>
</Viewpoint>
```

Figure 3: The Vehicle viewpoint in XML

```
<?xml version="1.0" ?>
<Viewpoint name="Gizmo">
  <Requirement id="1">
    The gizmo identifier is read by the system.
    <Requirement id="1.1">The gizmo identifier is validated by the
    system.</Requirement>
    <Requirement id="1.2">The gizmo is checked by the system for being
    active or not.</Requirement>
  </Requirement>
</Viewpoint>
```

Figure 4: The Gizmo viewpoint in XML

```
<?xml version="1.0" ?>
<Viewpoint name="TollGate">
  <Viewpoint name="PayingToll">
    <Requirement id="1">A green light is turned on if the gizmo is
    valid.</Requirement>
    <Requirement id="2">A yellow light is turned on if the gizmo is not
    present or invalid.</Requirement>
    <Requirement id="3">An alarm is sounded if the gizmo is not present or
    invalid.</Requirement>
  <Requirement id="4">
    The amount being debited is displayed if the gizmo is valid.
    <Requirement id="4.1">The amount being debited depends on the
    class of the vehicle.</Requirement>
  </Requirement>
  <Viewpoint name="SingleToll">
    <Requirement id="1">The amount being displayed is
    fixed.</Requirement>
  </Viewpoint>
  <Viewpoint name="ExitToll">
    <Requirement id="1">A yellow light is shown if the vehicle did
    not enter using a green lane.</Requirement>
    <Requirement id="2">The amount being debited depends upon
    the entry point.</Requirement>
  </Viewpoint>
</Viewpoint>
<Viewpoint name="EntryToll">
  <Requirement id="1">No signals are shown on passing an entry
  point.</Requirement>
</Viewpoint>
</Viewpoint>
```

Figure 5: The Toll Gate viewpoint in XML

### 4.1.2 Identify and specify concerns

*Concerns* are identified by analysing the initial requirements. For example, since the owner of a vehicle has to indicate his/her bank details during registration, *Security* is an issue that the system needs to address. Other *concerns* in our case study, identified in a similar way, are: *Response Time*, *Multi-Access System*, *Compatibility*, *Legal Issues*, *Correctness* and *Availability*. For simplification we choose to provide the specification of only three *concerns* here: *Compatibility*, *Response Time* and *Correctness* (figures 6 through 8). The choice is aimed at demonstrating a range of dimensions, between *Compatibility* and *Response Time*, and conflicts between *Response Time* and *Correctness*. The *Requirement* tag has the same semantics and scoping rules as for the viewpoints. The only difference is that the defining scope is now the *Concern*. Like viewpoints, *concerns* can be nested as well and *concern* names are unique with the scope of a case study in ARCADE.

```

<?xml version="1.0" ?>
<Concern name="Compatibility">
  <Requirement id="1">
    The system must be compatible with systems used to:
    <Requirement id="1.1">activate and reactivate gizmos;</Requirement>
    <Requirement id="1.2">deal with infraction incidents;</Requirement>
    <Requirement id="1.3">charge for usage.</Requirement>
  </Requirement>
</Concern>

```

**Figure 6:** The *Compatibility concern* in XML

```

<?xml version="1.0" ?>
<Concern name="ResponseTime">
  <Requirement id="1">
    The system needs to react in-time in order to:
    <Requirement id="1.1">read the gizmo identifier;</Requirement>
    <Requirement id="1.2">turn on the light (to green or yellow);</Requirement>
    <Requirement id="1.3">display the amount to be paid;</Requirement>
    <Requirement id="1.4">photograph the plate number from the rear;</Requirement>
    <Requirement id="1.5">sound the alarm;</Requirement>
    <Requirement id="1.6">respond to gizmo activation and reactivation.</Requirement>
  </Requirement>
</Concern>

```

**Figure 7:** The *Response Time concern* in XML

```

<?xml version="1.0" ?>
<Concern name="Correctness">
  <Requirement id="1">
    The system must ensure correctness of the data:
    <Requirement id="1.1">calculated within the system;</Requirement>
    <Requirement id="1.2">exchanged with the environment.</Requirement>
  </Requirement>
</Concern>

```

**Figure 8:** The *Correctness concern* in XML

### 4.1.3 Identify Coarse-grained Concern/Viewpoint Relationships

As we identify and describe viewpoints and *concerns* we can relate them, by building the matrix in table 3.

### 4.1.4 Identify candidate aspects

The matrix in table 3 shows which *concerns* cut across specific viewpoints. For example, we can observe that the requirements in the *Response Time concern* influence and constrain the

requirements in the viewpoints: *Gizmo*, *ATM*, *Toll Gate* and *Vehicle*. Similarly, *Compatibility* requirements crosscut the requirements specified by the *Police*, *Debiting System* and *ATM* viewpoints. In fact in our case study all the concerns are crosscutting. Consequently all the concerns identified form candidate aspects as they cut across multiple viewpoints. However, in another system a concern might constrain a single viewpoint and, hence, will not qualify as a candidate aspect (note that it will still be modularised as a *concern*).

Once a candidate aspect has been identified, the XML specification of the corresponding *concern* is transformed to reflect this fact. The transformation is a simple operation (using a simple transformation in XSLT – eXtensible Style Sheet Language for Transformations) which replaces the *Concern* tag with an *Aspect* tag. While this might seem a trivial transformation, it ensures that the specification reflects the aspectual nature of a *concern*.

### 4.1.5 Define composition rules

Composition rules define the relationships between aspectual requirements and viewpoint requirements at a fine granularity (unlike the relationship matrix in section 4.1.3 which is aimed at identifying candidate aspects). Composition rule definitions can be governed by an XML schema in ARCaDe. However, for simplification we describe the structure of composition rules with reference to some examples and not the XML schema definition. As shown in figures 9 and 10, a coherent set of composition rules is encapsulated in a *Composition* tag. Figure 9 encapsulates all compositions for the *Compatibility* requirements while figure 10 does so for *Response Time* requirements. The semantics of the *Requirement* tag here differ from the tags in the viewpoint and aspect definitions. Each *Requirement* tag has at least two attributes: the *aspect* or *viewpoint* it is defined in and an *id* which uniquely identifies it within its defining scope. If a viewpoint requirement has any sub-requirements these must be explicitly excluded or included in the *Constraint* imposed by an aspectual requirement. This is done by providing an *include* or *exclude* value to the optional *children* attribute. A value of *all* for a *viewpoint* or *id* value implies that all the viewpoints or requirements within the specified viewpoint are to be constrained.

**Table 3. Matrix relating concerns with viewpoints**

(**P:** Police; **Gz:** Gizmo; **DS:** Debiting System; **TG:** Toll Gate; **PT:** Paying Toll; **ST:** Single Toll; **ExT:** Exit Toll; **ET:** Entry Toll; **Vh:** Vehicle; **UV:** Unauthorized Vehicle; **VO:** Vehicle Owner; **Act:** Activation; **Reg:** Registration; **Bill:** Billing; **Adm:** Administration)

VP \ Concerns	P	Gz	DS	ATM	TG	PT	ST	ExT	ET	Vh	UV	VO	Reg.	Act.	Bill.	Adm.
Response Time		✓		✓	✓	✓	✓	✓	✓	✓	✓					
Availability		✓		✓	✓	✓	✓	✓	✓				✓	✓		✓
Security	✓		✓	✓								✓	✓	✓	✓	✓
Legal Issues	✓							✓					✓		✓	
Compatibility	✓		✓	✓										✓		
Correctness	✓	✓	✓		✓	✓	✓	✓	✓			✓	✓	✓	✓	
Multi Access		✓		✓	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓

The *Constraint* tag defines an, often concern-specific, action and operator defining how the viewpoint requirements are to be constrained by the specific aspectual requirement. Although the actions and operators are informal they must have clearly defined meaning and semantics to ensure valid composition of aspects and viewpoints. The *Outcome* tag defines the result of constraining the viewpoint requirements with an aspectual requirement. The action value describes whether another viewpoint requirement or a set of viewpoint requirements must be *satisfied* or merely the constraint specified has to be *fulfilled* (see table 6).

```
<?xml version="1.0" ?>
- <Composition>
  - <Requirement aspect="Compatibility" id="1.1">
    - <Constraint action="ensure" operator="with">
      <Requirement viewpoint="ATM" id="all" />
    </Constraint>
    <Outcome action="fulfilled" />
  </Requirement>
  - <Requirement aspect="Compatibility" id="1.2">
    - <Constraint action="ensure" operator="with">
      <Requirement viewpoint="Police" id="all" />
    </Constraint>
    <Outcome action="fulfilled" />
  </Requirement>
  - <Requirement aspect="Compatibility" id="1.3">
    - <Constraint action="ensure" operator="with">
      <Requirement viewpoint="DebitingSystem" id="all" />
    </Constraint>
    <Outcome action="fulfilled" />
  </Requirement>
</Composition>
```

**Figure 9:** The composition rules for *Compatibility* requirements

```
<?xml version="1.0" ?>
- <Composition>
  - <Requirement aspect="ResponseTime" id="1.1">
    - <Constraint action="enforce" operator="between">
      <Requirement viewpoint="Vehicle" id="1" />
      <Requirement viewpoint="Vehicle" id="2" />
    </Constraint>
    - <Outcome action="satisfied">
      <Requirement viewpoint="Gizmo" id="1" children="include" />
    </Outcome>
  </Requirement>
  - <Requirement aspect="ResponseTime" id="1.2">
    - <Constraint action="enforce" operator="between">
      <Requirement viewpoint="Gizmo" id="1" children="include" />
      <Requirement viewpoint="Vehicle" id="3" />
    </Constraint>
    - <Outcome action="satisfied" operator="XOR">
      <Requirement viewpoint="PayingToll" id="1" />
      <Requirement viewpoint="PayingToll" id="2" />
    </Outcome>
  </Requirement>
  - <Requirement aspect="ResponseTime" id="1.3">
    - <Constraint action="enforce" operator="between">
      <Requirement viewpoint="PayingToll" id="1" />
      <Requirement viewpoint="Vehicle" id="3" />
    </Constraint>
    - <Outcome action="satisfied">
      <Requirement viewpoint="PayingToll" id="4" children="include" />
    </Outcome>
  </Requirement>
  - <Requirement aspect="ResponseTime" id="1.4">
    - <Constraint action="enforce" operator="between">
      <Requirement viewpoint="PayingToll" id="2" />
      <Requirement viewpoint="Vehicle" id="4" />
    </Constraint>
    - <Outcome action="satisfied">
      <Requirement viewpoint="UnauthorisedVehicle" id="1" />
    </Outcome>
  </Requirement>
  - <Requirement aspect="ResponseTime" id="1.5">
    - <Constraint action="enforce" operator="between">
      <Requirement viewpoint="PayingToll" id="2" />
      <Requirement viewpoint="Vehicle" id="4" />
    </Constraint>
```

```
- <Outcome action="satisfied">
  <Requirement viewpoint="PayingToll" id="3" />
</Outcome>
</Requirement>
- <Requirement aspect="ResponseTime" id="1.6">
  - <Constraint action="enforce" operator="on">
    <Requirement viewpoint="ATM" id="1" children="include" />
  </Constraint>
  <Outcome action="fulfilled" />
</Requirement>
</Composition>
```

**Figure 10:** The composition rules for *Response Time* requirements

The informality of the actions and operators ensures that the composition specification is still readable by the stakeholders, an important consideration during requirements engineering. For example, if we look at the first composition rule in figure 10 and focus on the values in bold we get the following: “**Response Time** requirement **1.1** must be **enforced between** requirements **Vehicle 1** and **Vehicle 2** with the outcome that **Gizmo** requirement **1** including its children is **satisfied**”.

Tables 4, 5 and 6 describe the semantics of the actions and operators, which we have defined so far, for *Constraint* and *Outcome*. They also list the aspects that an operator or action might be specific to. The interesting point to note here is that not all operators are aspect-specific, e.g. XOR is a generic operator (reflected by a value ANY in the aspect column in table 4). Also, the actions for the *Outcome* are generic and not specific to a particular aspect. It is, however, not possible to say whether *Outcome* actions are always generic, as more case studies need to be carried out before arriving at such a conclusion. It is also worth noting (from table 5) that although the same operator might apply to different aspectual requirements, not all operator-action combinations are valid in the *Constraint* specification for a particular aspect. The combinations in table 5 apply to the case study in this paper only. More case studies need to be carried out to determine the complete set of valid operator-action combinations.

#### 4.1.6 Compose aspects and viewpoints

The aspects and viewpoints are composed using the composition rules. This leads to identification of conflicts among aspects whose requirements constrain the same or overlapping sets of viewpoint requirements. In case of ARCaDe this process is optimised as any potential interaction or conflict can be deduced from the composition rules. Consequently, one does not need to compose the aspects and viewpoints until the conflicts have been resolved. In another instantiation of the generic AORE model this might not be possible and composition might be required to identify conflicts.

#### 4.1.7 Handling conflicts

**Build the contribution table.** The contribution table shows in which way (negatively or positively) an aspect contributes to the others. The matrix represented in table 7 is symmetric, i.e. we only need to consider the diagonally upper triangle (or the lower one).

In this case, *Response Time* contributes negatively to *Security*, *Correctness* and *Multiple Access* and positively to *Availability*, for example. Whenever there is a negative contribution between aspects we are faced with conflict if these aspects apply to the same or overlapping sets of requirements in the viewpoints.

**Table 4:** Description of *Constraint* actions

Constraint Action		Aspects applicable to
Type	Description	
enforce	Used to impose an additional condition over a set of viewpoint requirements.	Response Time
ensure	Used to assert that a condition that should exist for a set of viewpoint requirements actually exists.	Availability, Compatibility, Correctness
provide	Used to specify additional features to be incorporated for a set of viewpoint requirements.	Security, Multiple Access
applied	Used to describe rules that apply to a set of viewpoint requirements and might alter their outcome.	Legal Issues
exclude	Used to exclude some viewpoints or requirements if the value <i>all</i> is specified.	ANY

**Table 5:** Description of *Constraint* operators

Constraint Operator		Action	Valid aspect: action-operator combinations
Type	Description		
during	Describes the temporal interval during which a set of requirements is being satisfied.	ensure	Availability: ensure-during
between	Describes the temporal interval falling between the satisfaction of two requirements. The interval starts when the first requirement is satisfied and ends when the second one is to start being satisfied.	enforce	Response Time: enforce-between
on	Describes the temporal point after a set of requirements has been satisfied.	enforce	Response-Time: enforce-on
for	Describes that additional features will complement the viewpoint requirements.	applied, provide	Legal Issues: applied-for Security: provide-for Multiple Access: provide-for
with	Describes that a condition will hold for two sets of requirements with respect to each other.	ensure	Compatibility: ensure-with
in	Describes that a condition will hold for a set of requirements that has been satisfied.	ensure	Correctness: ensure-in
XOR	Exclusive-OR (when either requirement is satisfied but not both)	ANY	ANY

**Table 6:** Description of *Outcome* actions

Outcome Action		Aspects applicable to
Type	Description	
satisfied	Used to assert that a set of viewpoint requirements will be satisfied after the constraints of an aspectual requirement have been applied.	ANY
fulfilled	Used to assert that the constraints of an aspectual requirement have been successfully imposed.	ANY

**Attribute weights to conflicting aspects.** We can help resolve aspectual conflicts by attributing weights to the cells of the aspect/viewpoint matrix (derived as the *concern*/viewpoint matrix in section 4.1.3) where the conflicting aspects apply to the same viewpoints. Weighting allows us to describe the extent to which an aspect may constrain a viewpoint (c.f. table 8).

The values are given according to the importance each aspect has for each viewpoint. The scales we are using are based on ideas from fuzzy logic and have the following meaning:

- *Very important* takes values in the interval ] 0,8 .. 1,0]
- *Important* takes values in the interval ] 0,5 .. 0,8]
- *Average* takes values in the interval ] 0,3 .. 0,5]
- *Not so important* takes values in the interval ] 0,1 .. 0,3]
- *Do not care* much takes values in the interval [0 .. 0,1]

Using fuzzy values (very important, important, not so important, etc) facilitates the stakeholders' task of attributing priorities to conflicting aspects. Therefore, for viewpoint *Gizmo*, for example, *Response Time* has higher priority than *Correctness* and *Multiple Access*, and *Correctness* has higher priority than *Multiple Access*.

**Resolve conflicts.** The conflict mentioned above should not be too difficult to resolve, as the weights express priorities. However, *Toll Gate* and its sub-viewpoints: *Paying Toll* (with sub-viewpoints: *Single Toll*, *Exit Toll*) and *Entry Toll* still show a conflicting situation between *Response Time* and *Correctness*. These two aspects contribute negatively to each other and have the same weight allocated to them (see the cells highlighted in table 8). Using ARCaDe we can determine that the conflict is in fact with reference to requirements 4 and 4.1 in *Paying Toll*. On one hand, the toll gate needs to react in time, on the other hand, it needs to display the correct amount. To resolve these kinds of

conflicts negotiation is needed among the stakeholders. One suitable solution will be to lower the weight allocated to *Response Time* to 0.8 for the affected viewpoints. This is because *Correctness* is more important than *Response Time*. It is essential that the correct amount is displayed (and subsequently

billed) even though the driver may not see it (if s/he is driving too fast).

Once all the conflicts have been resolved the specification is revised and recomposition carried out to identify any further conflicts.

**Table 7. Contribution matrix**

Aspects	Response Time	Availability	Security	Legal Issues	Compatibility	Correctness	Multi-Access
Response Time		+	-			-	-
Availability							+
Security						+	
Legal Issues					+	+	
Compatibility							
Correctness							
Multi-Access							

**Table 8. Matrix with weights to conflicting aspects**

(P: Police; Gz: Gizmo; DS: Debiting System; TG: Toll Gate; PT: Paying Toll; ST: Single Toll; ExT: Exit Toll; ET: Entry Toll; Vh: Vehicle; UV: Unauthorized Vehicle; VO: Vehicle Owner; Act: Activation; Reg: Registration; Bill: Billing; Adm: Administration)

VP	P	Gz	DS	ATM	TG	PT	ST	ExT	ET	Vh	UV	VO	Reg.	Act.	Bill.	Adm
Response Time		1,0		0,3	1,0	1,0	1,0	1,0	1,0	1,0	1,0					
Availability		✓		✓	✓	✓	✓	✓	✓				✓	✓		✓
Security	✓		✓	1,0									✓		✓	✓
Legal Issues	✓							✓							✓	
Compatibility	✓		✓	✓										✓		
Correctness	✓	0,8	✓		1,0	1,0	1,0	1,0	1,0			✓	✓	✓	✓	
Multi Access		0,3		0,3	0,3	0,3	0,3	0,3	0,3	0,3	0,3		✓	✓		✓

#### 4.1.8 Specify aspect dimensions

Specification of a candidate aspect's dimensions makes it possible to determine its influence on later development stages and identify its mapping onto a function, decision or aspect. Consider our *Compatibility* candidate aspect. The requirements derived from this aspect will influence parts of the system specification, architecture and design pertaining to requirements derived from viewpoints constrained by it. They will also influence system evolution as change of the user's ATM cards must be anticipated. The *Compatibility* aspect will, however, map on to a function allowing activation and reactivation of the gizmo. The *Response Time* aspect, on the other hand, will influence the type of architecture chosen and the design of the classes realising the requirements constrained by *Response Time*. It will map to an aspect at the design and implementation level because response time properties cannot be encapsulated in a single class and will be otherwise spread across a number of classes. The various candidate aspects in our case study and their mappings and influences are shown in table 9.

## 5. RELATED WORK

Recently there has been growing interest in propagating the aspect paradigm to the earlier activities of the software development lifecycle. A number of approaches to aspect-oriented design have been proposed e.g. [8, 9, 27]. At the Aspect-Oriented Software Development conference in 2002, one workshop was devoted to aspect-oriented requirements and architecture design [2].

Suzuki and Yamamoto propose an extension to UML to support aspects, where an aspect is described as a classifier in the meta-model [27]. The focus is on extending UML with aspects to support the design activity. An XML-based aspect description language is employed to interchange aspect models between development tools such as CASE tools and aspect weavers. The XML-based specifications supported by ARCaDe can also be used as an interchange mechanism.

Composition patterns [8, 9] is another approach to handle crosscutting concerns at the design level. This model, based on subject-oriented design [7] and UML, promotes reusability and traceability to the following activities of the software

development. In our approach the traceability of aspectual requirements is supported from the earlier stage of requirements engineering. This makes it possible to identify the mapping and influence of a requirements level aspect on the architecture and design in addition to the implementation.

**Table 9: Aspect dimension specification**

Candidate aspect	Influence	Mapping
Compatibility	Specification, architecture, design, evolution	Function
Response time	Specification, architecture, design	Aspect
Legal issues	Specification	Function
Correctness	Specification, design	Function
Security	Specification, architecture, design	Aspect
Availability	Architecture	Decision
Multi-user system	Architecture, design	Aspect

A UML compliant approach to handle quality attributes (i.e. non-functional requirements) at the early stages of the development process is proposed in [21]. Unlike the approach in this paper, separation of composition rules is not supported. Also, it is only possible to detect that a quality attribute affects a (set of) UML model(s). It is not possible to identify relationships at a more detailed level, such as between quality attributes and parts of the functionality represented in a UML model. The mapping and influence of a quality attribute on artefacts at later development stages is not discussed. Furthermore, the model in [21] does not provide explicit support for conflict resolution.

Dingwall-Smith and Finkelstein [11] present the building of a system for runtime monitoring of system goals. This system is specified using the KAOS approach [10] and currently it supports its *achieve pattern*. The major similarity with our work is in the way composition rules are separated. However, in their approach Hyper/J and its concepts are used as the definition language while we define a more abstract set of actions and operators that are independent of any aspect-oriented programming language. Also, while Dingwall-Smith and Finkelstein’s approach is specific to the construction of a system for runtime monitoring, we discuss modularisation and composition of aspects and stakeholders’ requirements at a general level, independently of any particular application or crosscutting concern.

In the Architecture Trade-off Analysis Method – ATAM [18] various competing quality attributes and their interactions are characterised. This is achieved by building and maintaining both quantitative and qualitative models of these attributes. The models are used as a basis to evaluate and evolve the architecture. The main focus of ATAM is on identifying the trade-off points at the architecture level. The work described in this paper focuses on identifying conflicting concerns and establishing critical trade-offs before the architecture is derived. Consequently, it is closer to the Twin Peaks model [22] which focuses on developing requirements and architectures in parallel in order to develop an early understanding of the system’s technical feasibility, discover further requirements and constraints and evaluate alternative design solutions.

The aspect-specific actions and operators employed by the composition rules bear a relationship with [23] which proposes employing the most suitable AOP technique to implement a particular concern. In our composition rules the most suitable action or operator is employed to deal with a particular requirements level aspect. This is also in close relation with the work on domain specific aspect modelling. One such environment and language is proposed by Gray et al. [14]. The focus of their work is on the design level while the composition rules, actions and operators discussed in this paper are aimed at the requirements level. The Embedded Constraint Language (ECL), based on OCL, employed by [14] is a design-level specification language that is not appropriate for requirements.

## 6. CONCLUSION AND FUTURE WORK

In the beginning we stated that the generic AORE model and its concrete realisation with viewpoints and XML is aimed as a stepping-stone towards two goals:

1. Providing improved support for separation of crosscutting functional and non-functional properties during requirements engineering hence offering a better means to identify and manage conflicts arising due to tangled representations;
2. Identifying the mapping and influence of requirements level aspects on artefacts at later development stages hence establishing critical trade-offs before the architecture is derived.

The separation of composition information from the aspects and viewpoints makes them highly independent of each other hence providing an improved separation of concerns. This also improves the reusability of the aspects in some instances. For example, the *Correctness* aspect in figure 8 is not specific to the toll collection system and may be reused to specify correctness requirements in another system. The *Availability* aspect (not shown in the paper) exhibits a similar degree of reusability. Since the composition rules operate at the granularity of individual requirements, it is possible to identify and manage conflicts at a fine granularity. This optimises the task of the requirements engineer identifying negotiation points for the stakeholders. Not only can the requirements engineer identify individual viewpoint and aspectual requirements for negotiation, s/he can also capture situations where there might be an apparent conflict with reference to the viewpoints, but no conflict at the level of individual requirements. The use of abstract, concern-specific operators coupled with the extensible model of XML ensures that the approach remains adaptable to other applications and extensible to incorporate new aspects as the requirements for the toll collection system change. The operators also help maintain the informality of the requirements specification while providing well-defined composition semantics.

The identification of the mapping and influence of a requirement level aspect promotes traceability of broadly scoped requirements and constraints throughout system development, maintenance and evolution. The improved modularisation and traceability obtained through early separation of crosscutting concerns can play a central role in building systems resilient to unanticipated changes hence meeting the adaptability needs of volatile domains such as banking, telecommunications and e-commerce.

With increasing support for aspects at the design and implementation level, the inclusion of aspects as fundamental modelling primitives at the requirements level and identification

of their mappings also helps to ensure homogeneity in an aspect-oriented software development process.

Our future work will focus on developing concrete realisations of the generic AORE model with use cases, scenarios and problem frames. We anticipate the extensible model of XML to play a fundamental role in this context (as has been the case for viewpoints). We intend to support these concrete models within ARCaDe. We also aim to develop support for validation of requirement level aspects. The case study in this paper contained only non-functional aspects. Another future direction will, therefore, be to conduct case studies involving both functional and non-functional aspects. We are also interested in exploring the use of fuzzy logic for trade-off analysis based on the weights we may give to aspects and viewpoints. This could help us identify a process to rank viewpoints and aspects by degree of importance in a system and use the result as a basis for incremental development.

**Acknowledgement:** This work is partly supported by CITI and ICCTI/British Council grants. The authors also wish to thank Peter Sawyer for helpful discussions during the course of this work.

## REFERENCES

- [1] "AspectJ Home Page", Xerox PARC, USA, <http://aspectj.org/>, 2002.
- [2] "Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design", Workshop at AOSD Conference, 2002, <http://trese.cs.utwente.nl/AOSD-EarlyAspectsWS/>.
- [3] "eXist XML Database Management System", W. Meier, <http://exist.sourceforge.net/>, 2002.
- [4] "XBel Framework - Reasoning with Inconsistency", A. Cheng, <http://www.cs.toronto.edu/~annie/FMGroup/xbel.html>, 2002.
- [5] L. Bergmans and M. Aksit, "Composing Crosscutting Concerns using Composition Filters", *CACM*, 44(10), 2001.
- [6] R. Clark and A. Moreira, "Constructing Formal Specifications from Informal Requirements", *Software Technology and Engineering Practice*, 1997, IEEE Computer Society Press, pp. 68-75.
- [7] S. Clarke, W. Harrison, H. Ossher, and P. L. Tarr, "Subject-Oriented Design: Towards Improved Alignment of Requirements, Design, and Code", *OOPSLA*, 1999, ACM, pp. 325-339.
- [8] S. Clarke and R. J. Walker, "Composition Patterns: An Approach to Designing Reusable Aspects", *ICSE*, 2001.
- [9] S. Clarke and R. J. Walker, "Towards a standard design language for AOSD", *AOSD*, 2002, ACM, pp. 113 - 119.
- [10] A. Dardenne, A. Lamsweerde, and S. Fickas, "Goal-directed Requirements Acquisition", *Science of Computer Programming*, Vol. 20, pp. 3-50, 1993.
- [11] A. Dingwall-Smith and A. Finkelstein, "From Requirements to Monitors by way of Aspects", *AOSD 2002 Workshop on Early Aspects*, 2002.
- [12] T. Elrad, R. Filman, and A. Bader (eds.), "Theme Section on Aspect-Oriented Programming", *CACM*, 44(10), 2001.
- [13] A. Finkelstein and I. Sommerville, "The Viewpoints FAQ." *BCS/IEEE Software Engineering Journal*, 11(1), 1996.
- [14] J. Gray, T. Bapty, S. Neema, and J. Tuck, "Handling Crosscutting Constraints in Domain-Specific Modeling", *CACM*, 44(10), pp. 87-93, 2001.
- [15] J. Grundy, "Aspect-Oriented Requirements Engineering for Component-based Software Systems", 4th IEEE Int'l Symp. on RE, 1999, IEEE CS Press, pp. 84-91.
- [16] M. Jackson, *Problem Frames: Analyzing and Structuring Software Development Problems*: Addison Wesley, 2000.
- [17] I. Jacobson, *Object-Oriented Software Engineering - a Use Case Driven Approach*: Addison-Wesley, 1992.
- [18] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The Architecture Tradeoff Analysis Method", *ICECCS*, 1998, IEEE CS Press, pp. 68-78.
- [19] A. Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour", 5th Int'l Symp. on RE, 2001, IEEE CS Press, pp. 249-261.
- [20] K. J. Lieberherr, D. Orleans, and J. Ovlinger, "Aspect-Oriented Programming with Adaptive Methods", *CACM*, 44(10), pp. 39-41, 2001.
- [21] A. Moreira, J. Araújo, and I. Brito, "Crosscutting Quality Attributes for Requirements Engineering", *SEKE*, 2002, ACM, pp. 167-174.
- [22] B. Nuseibeh, "Weaving Together Requirements and Architectures", *IEEE Computer*, 34(3), pp. 115-117, 2001.
- [23] A. Rashid, "A Hybrid Approach to Separation of Concerns: The Story of SADES", *Reflection conf.*, 2001, Springer-Verlag, LNCS 2192, pp. 231-249.
- [24] A. Rashid, P. Sawyer, A. Moreira, and J. Araujo, "Early Aspects: A Model for Aspect-Oriented Requirements Engineering", *IEEE Joint Int'l Conf. on RE*, 2002, IEEE CS Press, pp. 199-202.
- [25] G. Schneider and J. Winters, *Applying Use Cases - A Practical Guide*: Addison-Wesley, 1998.
- [26] I. Sommerville and P. Sawyer, *Requirements Engineering - A Good Practice Guide*: John Wiley and Sons, 1997.
- [27] J. Suzuki and Y. Yamamoto, "Extending UML with Aspects: Aspect Support in the Design Phase", *ECOOP Workshop on AOP*, 1999.
- [28] P. L. Tarr, H. Ossher, W. H. Harrison, and S. M. Sutton, "N Degrees of Separation: Multi-Dimensional Separation of Concerns", *ICSE*, 1999, ACM, pp. 107-119.