Analyzing Crosscutting in the Problem Frames Approach

M. Lencastre\textsuperscript{1}, J. Araujo\textsuperscript{2}, A. Moreira\textsuperscript{2}, J. Castro\textsuperscript{3,4,γ}

\textsuperscript{1}Departamento de Computação, Universidade de Pernambuco
Rua Benfica, 455 Madalena, 50750-410, Recife – PE, Brazil
maria@dsc.upe.br

\textsuperscript{2}CITI/Departamento de Informática, FCT, Universidade Nova de Lisboa
Quinta da Torre, 2829-516, Caparica, Portugal
\{ja, amm\}@di.fct.unl.pt

\textsuperscript{3}Centro de Informática, Universidade Federal de Pernambuco, Cidade Universitária, 50750-410, Recife, Brazil
jbc@cin.ufpe.br

\textsuperscript{4}Istituto Trentino di Cultura - ITC
Istituto per la Ricerca Scientifica e Tecnologica - IRST, Trento-Povo, Itália
jaelson@itc.it

Abstract
The main focus of the Problem Frames approach is on improving domain and business modeling, as well as requirements engineering. Until now the Problem Frames approach did not explore in depth the crosscutting nature of some properties of a problem, leaving out the treatment of global properties that affect the whole or part of a system, and which might have a negative impact on the modularization of concepts and therefore in the systems’ evolution. On the other hand, aspect-oriented software development offers techniques to handle crosscutting concerns in a systematic and effective way, offering improved support for software maintenance and evolution. This paper integrates aspect concepts into Problem Frames.

Categories and Subject Descriptors
Software; Software Engineering; Requirements/ Specifications.

General Terms:
Design.

Keywords: requirements, aspects, problem frames, crosscutting elements.

1. Introduction
Aspect-Oriented Software Development (AOSD) aims at handling crosscutting concerns by proposing means to their systematic identification, modularization and composition [4]. Crosscutting concerns are properties whose implementation is scattered among several implementation modules, producing tangled systems that are difficult to understand, maintain and evolve.

In spite of the proven importance of handling crosscutting concerns (i.e., aspects) early in the software development life-cycle [2, 3, 4, 8, 10, 11], modern approaches such as Problem Frames [12, 14, 15] have not yet considered these features in depth. The Problem Frames approach focuses on the early life-cycle software engineering, emphasizing the problem to be solved rather than the solution defined for the problem [14]. It groups problems by type; they are similar to design patterns – elements of reusable object-oriented software [8], but they are problem-oriented rather than solution-oriented. Problem frames make it easier to solve a problem once the type of a problem has been classified. Problem Frames incorporate relevant concepts such as domains, phenomena, problem frames, and concerns (e.g. main and particular concerns). Like use cases [9], Problem Frames are mainly used to define functional requirements; non functional requirements are not the focus of the approach. Moreover, we have noticed that the crosscutting nature of requirements is not explored effectively in the Problem Frames approach.

The objective of this paper is to extend the Problem Frames meta-model proposed in [16] to include the existing crosscutting relationships between the main elements. This extension provides the identification of classes of candidate aspects in the Problem Frames approach, which can be further investigated.

This paper is organized in the following way. Section 2 introduces some basic concepts of both AOSD and Problem Frames. Section 3 illustrates Problem Frames by means of an example. Section 4 discusses a meta-model for Problem Frames with aspects. Section 5 presents some related work.
Finally, section 6 draws some conclusions and points out directions for possible future work.

2. Background

2.1. Aspect-Oriented Software Development

AOSD [18] addresses separation of concerns by offering means to decompose a system into distinct parts that overlap in functionality as little as possible. In particular, AOSD focuses on the modularization and encapsulation of crosscutting concerns. The term crosscutting concerns refers to properties of software that cannot be effectively modularized using traditional software development techniques, such as object-oriented methods. Notice that “crosscutting” is a relationship between concerns. When we say that a concern is crosscutting we are implicitly acknowledging some dominant decomposition that offers the base over which the crosscutting concern cuts across. Typical examples of crosscutting concerns are non-functional requirements, such as security, fault tolerance, persistence. However, crosscutting concerns can also be functional requirements, such as auditing or data validation [3, 7, 10].

Crosscutting concerns are encapsulated in separate modules, known as aspects, and composition mechanisms are later used to weave them back with other core modules, at loading time, compilation time, or run-time. However, aspects, as well as their compositions, also have an important role to play before the implementation activity. Aspects allow the modularization of crosscutting concerns that cannot be encapsulated by a single use case [9], or viewpoint [1], for example, and are typically spread across several of them. Composition, on the other hand, apart from allowing the developers to picture a broader part of the system, allows them to identify conflicting situations whenever a concern contributes negatively to others. This offers the opportunity to establish critical trade-offs before the architecture design is derived, supporting the necessary negotiations among the stakeholders [4].

AOSD aims at addressing such crosscutting concerns at the various levels of the software development process, by providing means for their systematic identification, separation, representation and composition [4].

2.2 Problem Frames Overview

Problem Frames [14] can be seen as a software engineering technique appropriate for describing problem domains, defining requirements and decomposing the problem into sub-problems. This technique clearly separates (i) the description of the problem without the solution, and (ii) the description of the problem with the solution. By “solution” we mean what is desired and planned to solve the problem. Both of the descriptions are based on domains, which are composed of phenomena, and their interactions. The main concepts of the approach are taken from [14] and are detailed next.

A domain can be thought of as a collection of related phenomena (such as entities, events and states), which represent part of a physical domain in the world where the effects are observable. The only way two domains can interact is through a connection between them, which is called interface of shared phenomena. A domain that must be designed and built, in the form of software, to solve the problem is called a machine domain. A problem details the problem context, including explicitly the problem requirement and how the requirement is related to its domains (constraints or references the domain’s phenomena), and the roles that the domains play in the problem (controlled or observed). Concerns are characteristics of a problem demanding the developer’s attention. Generally, a problem has a main concern (the frame concern) and also other concerns.

A problem frame is a kind of pattern that captures and defines commonly found classes of simple sub-problems. It defines an intuitively identifiable problem class in terms of its context and characteristics of its domains, interfaces and requirements. Jackson supplies a repertoire of recognized problem classes (problem frames). In this paper we use Problem Frames, in uppercase, to reference the approach and problem frames, in lowercase, to reference problem classes.

3 Problem Frames: an Illustrative Example

This section illustrates the problem frames main concepts by means of a small car park example. This example will be the used to describe crosscutting relationships between problem frames concepts. This example was taken from [11, 16]. The problem context for the car parking control system can be described as follows:

“To use a car parking system, a client has to get a ticket from a machine after pressing a button. Afterwards, the car is allowed to enter and park in an available place. The system has to control if the car parking is full or if it still has places left. When s/he wants to leave the parking place, s/he has to pay the ticket obtained (described above) in a paying machine. The amount depends on the time spent. After paying, the client can leave by inserting the ticket into a machine, which will open the gate for her/him to leave. Regular users of the parking system may pre-purchase time and enter/exit by inserting a card and PIN number which will result in money being deducted automatically from the user’s card (which has an identification number and a pin number)".

As described in [16], the problem of the car parking control can be structured and analyzed by the context diagram presented in Figure 1. It includes domains which consider information such as how many cars can enter, how to pay the parking, how to control the user’s accounts, how to control the cars’ entrance and exit, and so on.
The context diagram structures and delimits the problem as a number of separated domains (Park, Entry Machine, Pay Machine, Exit Machine, Park Display, and Driver), together with the machine to be built (the Car Park Control). It shows how these domains interact with each other and with the machine through the interface of shared phenomena, such as the ones presented in Figure 1:

- a: parking information;
- b: card number for regular users, register entrance data, ticket number;
- c: card number, ticket number, information checked, ex: grace period from paying ticket exceeded entry not liberated;
- d: register entrance, calculated cost;
- e: card number for regular users;
- f: state of the park;
- g, h, i: interaction not treated by the car parking control system, like the interaction between the driver and the machines (i.e. press button of an Entry machine to get a ticket, etc).

Considering the problem context of the example, some concerns can be identified, such as the need for:

- Monitoring of machines: i.e. access to information about the number of vacant spaces, issued tickets, paid values, cars entering and exiting, valid cards and tickets;
- Validate card numbers and ticket numbers;
- Monitoring machines to determine whether they are functioning properly;
- Maintenance of parking data (time/money/users): existing card numbers, register entry information and paid values, number of places, number of occupied places;
- Control the display about park state (i.e. “no space in park”);
- Calculation of parking cost (amount to be paid), based on spent time;
- Calculate exceeded time/cost in case of exceeding grace period;
- Assure the payment of cars parking;
- Allow car entrance even if there is no space, to obtain the maximum occupation of places;
- Security in pin card numbers.

Notice that it is not the aim of this paper to show how aspects are identified in an automatic fashion. Nevertheless, some approaches can be used for that purpose, such as [5, 7].

As described in [16], the car parking control problem can be further decomposed into the sub-problems:

1. **Park Monitor (PMM)**: this controls the availability of places and warns park “full” or “vacant”;
2. **Machine Monitor (MMM)**, which can be decomposed into:
   - **Enter Machine Monitor (EMM)**: this checks the machine state (broken gate, lack of paper, off, etc.); and displays machine state;
   - **Exit Machine Monitor (XMM)**: this checks the machine state (broken gate, off, etc.); and displays machine state;
   - **Pay Machine Monitor (PMM)**: this checks the machine state (broken, out of money, out of paper etc.); and displays the machine state;
3. **Park Use Control (PUCM)**, which can be decomposed into:
   - **Entry Authorization (EAM)**: this checks free places, registers entrance (register ticket/card number, ticket/card date time), and sets new occupation place;
   - **Pay Control (PCM)**: this validates car entrance, calculates time spent, informs amount to be paid, registers payment value and ticket/card; calculates exceeded cost and time in case of expiration of grace period.
   - **Exit Authorization (XAM)**: this validates payment; registers exit card/ticket; sets new exit; sets new vacant place.

4. **A Meta-model for Problem Frames with Crosscutting Relationships**

Our first attempt to define the Problem Frames concepts through a meta-model can be found in [16]. This paper extends [16] in two ways: (i) identifying some main model elements that crosscut other ones and (ii) extending that meta-model to consider the crosscutting nature of some problem frames concepts. As we will see, some problem frames model elements crosscut other model elements. These relations can be represented through crosscuts at the meta-model level. Figure 2 presents part of the meta-model, taken from [16], changed to consider the identified crosscutting relationships.
If a domain affects several problems we can say that it is a crosscutting or aspectual domain. Likewise, if a requirement crosscuts several domains it is a crosscutting or aspectual requirement. We can notice that, in the proposed meta-model extension, all the classes crosscut each other: Problem, Requirement, Phenomena, Domain, Concern, and Problem Frames.

A model, with so many crosscutting relationships, introduces a new level of detail for system evolution and for improving system traceability. However, it also increases the model complexity, requiring an implementation tool to facilitate its use.

To illustrate these crosscutting concepts in the car parking example, we use the following strategy. Considering problems and requirements that compose the main problem of car parking control, we identify all the related data which have crosscutting relationship with these problems (see Table 2). Then we specify and exemplify some other crosscutting relationships between Concerns and Domains (Table 3), and between Domains and Phenomena (Table 4).

The goal here is not to make an exhaustive description of the example, but only to show the scattered and duplicated data among the model elements.

Figure 2. A meta-model for an aspect-oriented version of problem frames
4.1 Crosscutting Relations with Problems
Table 2 presents the model concepts (row 1) that crosscut the car park problems (column 1):
- ValidateCard/Ticket requirement crosscuts many problems, such as EAM, PCM and XAM;
- The domain Entry Machine crosscuts many problems such as MMM and EAM;
- The basic problem InfoDisplay is applied in many problems such as PMM and MMM.
- The TicketNum is a phenomena that crosscuts many problems such as PCM, XAM and EAM;
- Many concerns crosscut different problems such as Persistence and Security in EAM, PCM, XMA. Here concerns include both main frame concerns and particular concerns.

In Table 2, the problems represent the machine domains. To simplify, in the column Domain we do not include the involved machine domains.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Requirement</th>
<th>Phenomenon</th>
<th>Domain</th>
<th>Basic PF</th>
<th>Concern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Park Monitor (PMM)</td>
<td>ControlDisplay/About</td>
<td>NumPlaces Full, Vacant</td>
<td>ParkDisplay</td>
<td>InfoDisplay SimWorkPiece</td>
<td>DisplayPark Information</td>
</tr>
<tr>
<td></td>
<td>ParkState</td>
<td>NumOccupiedPlaces</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MonitorPayMachine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MonitorExitMachine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EntryAuthorization (EAM)</td>
<td>RegisterCarEntrance Ticket or Card</td>
<td>RegisterEntryNumOccupiedPlaces CardNum, TicketNum</td>
<td>ParkInfo</td>
<td>EntryMachine ReqBehavior SimWorkPiece Transformation</td>
<td>Persistence Security ValidateDriver</td>
</tr>
<tr>
<td>PayControl (PCM)</td>
<td>GetRegisterEntrance</td>
<td>RegisterEntrance CardNum TicketNum TimeSpent CalculatedAmount ExceededTime cost</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CalculateTimeSpent</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CalculateAmountToBePaid</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>RegisterPayment</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.2. Crosscutting Relations with Domains
Table 3 shows some examples of phenomena which cut across different domains, i.e. aspectual phenomena. For example, RegisterEntrance crosscuts the domains EM, PM, XM and P (park information).

Table 3. Crosscutting of “Phenomenon vs. Domain”

<table>
<thead>
<tr>
<th>Phenomenon</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegisterEntrance</td>
<td>EM, PM, XM, P</td>
</tr>
<tr>
<td>ExceededTime</td>
<td>PCM, XAM, P</td>
</tr>
<tr>
<td>NumOccupiedPlaces</td>
<td>P, EM, XM, X</td>
</tr>
<tr>
<td>CardNum</td>
<td>EAM, PCM, XAM, P</td>
</tr>
</tbody>
</table>

4.3. Crosscutting Relations with Concerns
Table 4 shows some examples of concerns that cut across existing domains. For example, Machine Monitoring crosscuts the domains PMM, EM, PM and XM.

Table 4. Crosscutting of “Concern vs. Domain”

<table>
<thead>
<tr>
<th>Concern</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>MachineMonitoring</td>
<td>PMM, EM, PM, XM</td>
</tr>
<tr>
<td>Persistency</td>
<td>EAM, PCM, XAM, P</td>
</tr>
<tr>
<td>Security</td>
<td>EAM, PCM, XAM</td>
</tr>
<tr>
<td>DisplayPark Information</td>
<td>PMM, MMM</td>
</tr>
</tbody>
</table>

4.4. Discussion
The Problem Frames approach has demonstrated that it includes many model elements (e.g. Persistency, Security, ValidateCard/Ticket) that can be seen as aspects as they affect several others. But why is this important to take into consideration? The main advantage is to be able to enhance the modularization of the crosscutting models element, be requirements, concerns, domains, phenomena or basic problem frames. This, in turn, can be used to derive a better architecture design that consequently provides a more robust
implementation [19]. Moreover, identifying and managing aspects at early stages help detect conflicting concerns early, when trade-offs can be resolved more economically, as discussed in [4].

In summary, the aspects, their relationships and classification seems to be a promising way for exploring the power of aspects techniques.

5. Related Work
In spite of several searches, there are not many works handling problems frames and aspects. In [6] the authors derive security requirements from crosscutting threat descriptions using the Problem Frames approach. In spite of the relevance of this work, it does not present an explicit way to deal with aspects in a general fashion and also does not include the main model elements’ analysis.

In [16] a meta-model for Problem Frames was presented with the aim to improve its understanding and give support for the development of a specific tool and mapping between other techniques, such i* and other approaches which deal with early aspects.

6. Conclusions and Future Work
We have explored how to identify crosscutting relationships between problem frames concepts. This was achieved by extending the problem frames’ meta-model presented in [16], by including crosscutting relationships between model elements. This study was illustrated with an example.

We believe that the modeling power of problem frames is augmented by exploring crosscutting relationships that may exist between Problem Frames model elements. The result is the production of models more capable of handling requirements evolution, exploring aspects and improving system traceability.

Currently we are studying how to extend the Problem Frames notation to highlight aspectual model elements, as well as developing more examples to demonstrate the advantages of this extension. For future work, we plan to investigate crosscutting between other concepts such as frame variants and flavors, and also to develop a tool to support a hybrid approach [17].

7 Acknowledgements
This work has been supported by research projects: CAPES/GRICES 129/05, CAPES/CPLP 4.1.3, CAPES BEX 1775/2005-7 and CNPq 304982/2002-4.

References