

A Taxonomy of Asymmetric Requirements Aspects

Nan Niu[†], Steve Easterbrook[†], and Yijun Yu[‡]

[†]Department of Computer Science, University of Toronto
Toronto, Ontario, Canada M5S 3G4
{nn,sme}@cs.toronto.edu

[‡]Computing Department, The Open University
Walton Hall, Milton Keynes, UK MK7 6AA
y.yu@open.ac.uk

Abstract. The early aspects community has received increasing attention among researchers and practitioners, and has grown a set of meaningful terminology and concepts in recent years, including the notion of *requirements aspects*. Aspects at the requirements level present stakeholder concerns that crosscut the problem domain, with the potential for a broad impact on questions of scoping, prioritization, and architectural design. Although many existing requirements engineering approaches advocate and advertise an integral support of early aspects analysis, one challenge is that the notion of a requirements aspect is not yet well established to efficaciously serve the community. Instead of defining the term once and for all in a normally arduous and unproductive conceptual unification stage, we present a preliminary taxonomy based on the literature survey to show the different features of an asymmetric requirements aspect. Existing approaches that handle requirements aspects are compared and classified according to the proposed taxonomy. In addition, we study crosscutting security requirements to exemplify the taxonomy’s use, substantiate its value, and explore its future directions.

1 Introduction

The many early aspects researchers and practitioners have grown a set of meaningful terminology and concepts, of which the most prominent is the distinction made between code-level and early aspects. Early aspects are concerns that crosscut an artifact’s dominant decomposition, or base modules derived from the dominant separation-of-concerns criterion, in the early stages of the software life cycle [5]. “Early” signifies occurring before implementation in any development iteration, and embodies the key activities of requirements engineering, domain analysis, and architecture design, as indicated in the early aspects Web portal [12].

It is probably not a coincidence that one of the earliest descriptions of early aspects appeared in the proceedings of the premier requirements engineering (RE) conference in 1999 [14]. And the fact that the most recent RE conference

(RE'06) presented the best paper award to an early aspects paper [17] demonstrates the strong connection between RE and early aspects communities. In fact, many existing RE approaches advocate and advertise an integral support of early aspects analysis, e.g., use cases [20], scenarios [3], viewpoints [28], and goals [36].

Aspects at the requirements level present stakeholder concerns that crosscut the problem domain, with the potential for a broad impact on questions of scoping, prioritization, and architectural design. A thorough analysis of early aspects in requirements offers a number of benefits:

- Explicit reasoning about interdependencies between stakeholder goals and constraints;
- Improving the modularity of the requirements structure;
- Identification of the impact of early aspects on design decisions can improve the quality of the overall architectural design and implementation;
- Conflicting concerns can be detected early and trade-offs can be resolved more economically;
- Test cases can be derived from early aspects to enhance stakeholder satisfaction; and
- Tracing stakeholder interests throughout software life cycle becomes easier since crosscutting concerns are captured early on.

Despite the growing awareness of these benefits and the continuing endeavor to achieve them, one challenge RE and early aspects communities currently face is that the notion of a requirements aspect, i.e., aspect at the requirements level, is not yet well established. We seek to explain and clarify the requirements aspects phenomena. The goal is to provide requirements analysts and other stakeholders a foundation for discussing specific challenges they might face in aspect-oriented RE projects. To this end, we conduct literature survey and domain analysis, thus presenting a taxonomy to show the different features of a requirements aspect. The sources of our survey focus mainly on asymmetric approaches and come primarily from the publications and reports in the literature, which can be found in [12].

Reference models with the unifying taxonomy represent prototypical models of some application domain, and have a time-honored status. One well-known example is the OSI 7-layer reference model, which divides network protocols into seven layers: physical, data link, network, transport, session, presentation, and application. The taxonomy of protocol layers is in widespread use, and is discussed in virtually every basic textbook on computer networks. The OSI 7-layer model is successful because it draws on what was already understood about the networks domain, thus codifying the core knowledge to be flexible.

Not every domain is sufficiently standardized to allow for a reference model or a unified taxonomy. In a blooming research field such as early aspects, people explore and tackle recognized problems complementarily, based on different mindsets and traditions. Various perspectives may not converge in a short period of time, which makes the conceptual unification process arduous and unproductive most of the time. As an example, the term “component” is used to describe

rather different concepts in software engineering: subsystems, JavaBeans, ActiveX controls, .NET assemblies, CORBA components, and more. As we shall see in section 2, the term “requirements aspect” saliently resembles in this respect the notion of a component.

Moreover, approaches like the one followed by the OSI 7-layer model are not necessarily supportive of change, especially when this change goes beyond the initially covered domain. However, at least experience on how to get to the model and the taxonomy should be recorded and shared [29], as undeniably a contribution to knowledge. Along this line, we present an initial attempt to identify reusable resources in the domain of requirements aspects. In section 3, We document our domain analysis results – a taxonomy of requirements aspects – using feature diagrams [21], and then apply the taxonomy to compare existing approaches that deal with aspects at the requirements level.

To exemplify the taxonomy’s use and substantiate its value, in section 4, we present a reified requirements aspect – security, one of the most mentioned crosscutting concerns in the current literature. We then review related work in section 5, and conclude the paper with a summary and some directions for future work in section 6.

2 A Teaser Description for Requirements Aspects

There exist several definitions and descriptions for the term “requirements aspect”, one of them by the initiators of research in early aspects: Elisa Baniassad, Paul Clements, João Araújo, Ana Moreira, Awais Rashid, and Bedir Tekin-erdoğan:

“A requirements aspect, then, is a concern that cuts across other requirement-level concerns or artifacts of the author’s chosen organization. It is broadly scoped in that it’s found in and has an (implicit or explicit) impact on more than one requirement artifact.” [5]

We will use it as a starting point for our further discussion. Let’s consider some parts of this description in detail:

- “a concern”: Calling a requirements aspect a concern develops strong ties with stakeholders of the intended software system. A requirements concern is a matter of relevance that conveys the problem domain’s property of interest to specific stakeholders. Therefore, any requirements aspect has its intent or purpose of existence, and needs to be traced to some stakeholder interest. Addressing requirements concerns thus enhances the stakeholder satisfaction and the overall software quality.
- “cuts across”: No matter how stakeholder concerns are structured, a requirements aspect crosscuts the dominant decomposition, i.e., the author’s chosen organization. This view assumes that the author has chosen some dominant organizing decomposition structure at the requirements level first, and makes the crosscutting concern a second-class object. Actually, aspect has become an equivalent substitute for a crosscutting concern in the literature.

- “author’s chosen organization”: Current requirements techniques offer a variety of structures for organizing the requirements, such as (structured) natural languages, use cases, scenarios, viewpoints, goal models, features, etc. [26]. The requirements analyst (author) develops a relatively well-organized set of requirements based on some dominant decomposition criteria and chosen structures. Requirements aspects emerge as a result of the lack of additional decomposition dimensions of the author’s chosen organization.
- “broadly scoped”: The authors of the original article [5] stated that broadly scoped properties can be quality attributes (nonfunctional requirements) as well as functional concerns that the requirements engineer must describe with relation to other concerns. These broadly scoped properties manifest as scattered and tangled concerns in the author’s chosen organization of requirements.
- “impact on”: Requirements aspects do not exist in isolation. They need to contact other requirement-level concerns or artifacts to provide some service according to their purpose of existence. This service providing process, which is also known as aspect weaving, is *usually* done in an oblivious fashion, i.e., the service provider (aspect) is impelled toward the consumers without them being aware of aspect’s existence. Recent work has pointed out that obliviousness is neither an essential nor a desirable property of aspects [32]. Nevertheless, aspects need to explicitly specify the conditions, locations, and implications of the intended interactions.

This description of the term “requirements aspect” is very generic and thus it is not surprising that the term is used to describe rather different concepts: a collaboration in requirements for software components [14], an extension in a use case diagram [20], a softgoal in a goal model [36], an instance of terminological interference in viewpoints-based requirements models [25], a non-functional requirement (NFR) in a software requirements specification [17], and more.

The purpose of this paper is to try to clarify these different views by providing a taxonomy for requirements aspects. We therefore do not try to give one concise, closed definition. Instead, we will show the different features and characteristics a requirements aspect must or can have, thereby classifying the different kinds of requirements aspects as they are used today.

3 A Feature Diagram for Requirements Aspects

This section presents the results of applying domain analysis to existing approaches that tackle requirements aspects. Domain analysis is concerned with analyzing and modeling the variabilities and commonalities of systems or concepts in a given domain, thereby developing and maintaining an information infrastructure to support reuse [11].

We document our domain analysis results – a taxonomy of requirements aspects – using feature diagrams [21]. Essentially, a feature diagram is a hierarchy

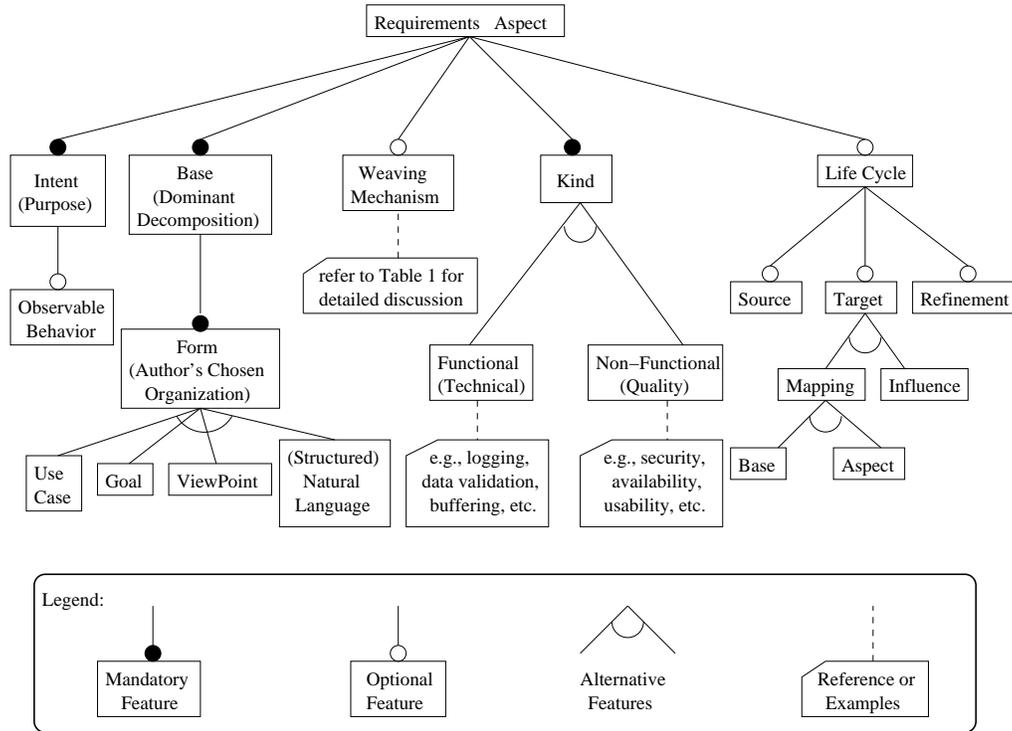


Fig. 1. A feature diagram for asymmetric requirements aspects.

of common and variable features characterizing the set of instances of a concept [9]. In our case, the features provide a taxonomy and representation of design choices for approaches dealing with aspects at the requirements level.

We do not aim for this taxonomy to be normative and immune from change. In fact, the relatively new area of early aspects has already experienced many overloaded terms, and many of the terms we use in our taxonomy are often used differently in descriptions of other approaches. Consequently, we provide minute examinations of the terms and concepts as we use them. Furthermore, we expect the taxonomy to evolve as our understanding of requirements aspects matures. Our main goal is to show the vast range of available choices as represented by the current approaches, from a reuse perspective.

Figure 1 depicts a feature diagram we use as a basis for our discussion on requirements aspects. We deliberately restrict the diagrammatic notations in Fig. 1 to conforming with those originally proposed in [21], with reference and annotated examples provided to illustrate specific concepts. In particular, only three types of lines (edges) connecting boxes (nodes) are presented: mandatory, optional, and alternative (XOR-choice). This is because the notations in [21] are free of ambiguities, whereas later modified feature diagram variants are neither precise nor ambiguity-free [33]. While we will clarify the diagram semantics on

the fly, we refer the interested reader to [21, 9, 33] for detailed discussion on feature modeling.

In Fig. 1, the features (denoted by the boxes) of the concept *requirements aspect* are described, which is located at the top of the feature diagram. The boxes directly connected to *requirements aspect* are the direct sub-features of a requirements aspect. The little circles at the edges connecting the features define the semantics of the edge. A filled circle means mandatory. Thus, every requirement aspect has an *intent* or *purpose* of existence, is of a certain *kind*, and cuts across multiple *base* modules according to the *dominant decomposition*. Optionally (denoted by the outlined circle at the edge), a requirements aspect provides *life cycle* information, and defines some *weaving mechanism*. Alternative features means an exclusive-or choice. For example, analysts organize requirements using one and only one base form in a particular RE stage like use cases, goals, viewpoints, and so forth.

We elaborate sub-features presented in Fig. 1 in the following sub-sections, and apply the taxonomy to compare different requirements aspects approaches. As long as we organize the discussion by (sub-)features, “aspects” emerge and crosscut this dominant decomposition structure. We therefore discuss these cross-cutting properties in a separate sub-section.

3.1 Intent and Life Cycle

An aspect represents some concern, which only matters to specific stakeholders of a software system. A requirements aspect represents stakeholder interest in the problem domain, including high-level concerns like user satisfaction and happiness, system qualities like security and efficiency, software capabilities like fault tolerance and persistent storage, overall considerations like development time and return on investment, and many others. Each requirements aspect, therefore, needs to have an intent to help justify its very existence with particular stakeholder interest.

In many cases, the intent is further formulated and elaborated through observable behaviors to reflect what the stakeholder has in mind to expect the software to do or bring about. For example, to ensure security, the software system shall disable user accounts after the wrong password is entered three times. These observable behaviors are what make the intent operationalizable and measurable. Naturally, test cases can be derived to check whether the resulting software meets the intent, addresses the stakeholder concern, and guarantees the desired behavior.

Test cases present an instance of the *life cycle* feature, which supports traceability of requirements aspects throughout the software development life cycle. Requirements traceability is defined by Gotel and Finkelstein [13] as the ability to describe and follow the life of a requirement in both a forwards and backwards direction, i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of on-going refinement and iteration in any of these phases.

Tracing a requirements aspect backwards to its origins helps uncover its intent and the source of stakeholder interest: whether the aspect is a concern relating to business, technology, organization, process, etc. From the standpoint of aspect-oriented program analysis and evolution, this source information about requirements aspects provides a baseline to validate code aspects against stakeholder relevance: are they merely refactored based on particular implementations or originally required by specific stakeholders?

Tracing a requirements aspect forwards to its subsequent development involves systematic decision making. An aspect identified at the requirements level can be designed as a coherent module conforming with the author's chosen dominant organization structure. For example, synchronization can be mapped to a single class in an object-oriented implementation. Alternatively, a requirements aspect can be mapped to an architectural-, code-, or test-level aspect, following the aspect-oriented software development paradigm. This preserves the concern's crosscutting nature throughout the software development life cycle. Other than being modularized definitively in design and implementation, a requirements aspect can be recorded as a tentative issue to influence ensuing development and decision making.

This tentative tracing strategy allows requirements aspects to be refined so that nobody is forced to make premature design decisions. Refinement supports iterations of analysis and interactions with stakeholders. The goal is to elaborate requirements and architecture in parallel within the system context,¹ better determine what crosscuts what, and tease out the implications of the identified crosscuttings.

3.2 Base and Weaving

Traditional waterfall development process produces artificially frozen requirements documents for use in the next step in the development life cycle. Alternatively, this process creates systems with constrained architectures that restrict users and handicap developers by resisting inevitable and desirable changes in requirements. It would be imprudent to assume that one could fully understand and document system requirements up-front. In practice, key requirements are not fully understood until the architecture is baselined at the conclusion of the refinement and elaboration phase.

The Twin Peaks model presented by Bashar Nuseibeh [27] vividly depicts this iterative and progressive process, and strongly emphasizes the concurrent interplay between requirements and architecture: candidate architecture can constrain designers from meeting particular requirements, and the choice of requirements can influence the architecture that designers select or develop [27]. Although the Twin Peaks model develops requirements and architectural specifications concurrently, it continues to separate problem structure and specification from solution structure and specification, in an iterative process that produces

¹ We will discuss the Twin Peaks model of weaving together requirements and architecture [27] in more detail in the next sub-section.

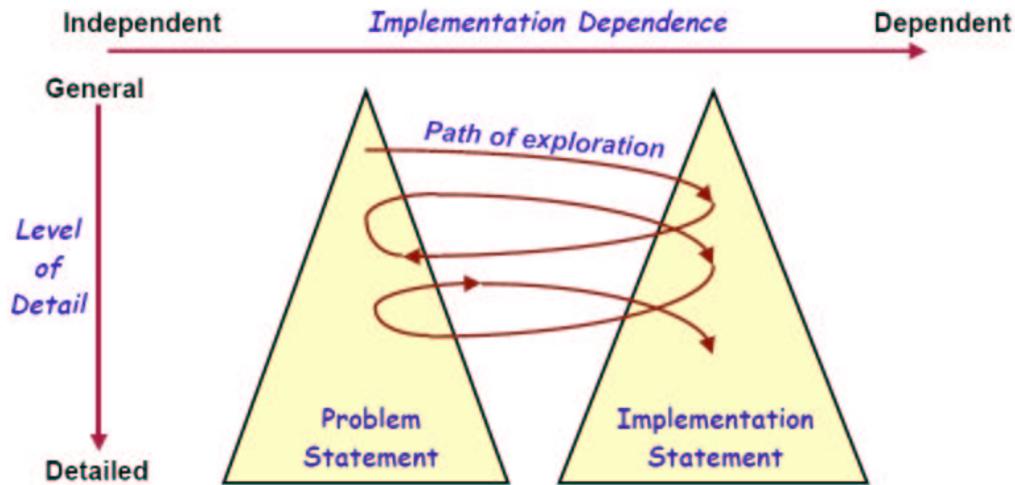


Fig. 2. The Twin Peaks model (*Adpated from [27]*).

progressively more detailed requirements and design specifications, as Fig. 2 suggests.

The Twin Peaks model further suggests an agenda “from early aspects to late requirements”, because identifying aspects too early is counterproductive [28]. Following this school of thought, we can safely assume an asymmetric, base-aspect separation approach to tackling requirements aspects. That is, progressively exploring crosscutting aspects is built upon a relatively well-structured set of requirements of the author’s chosen base organization, with the form varies over a wide range: such as (structured) natural languages, use cases, viewpoints, goal models, etc. [26].

On the contrary, a symmetric approach does not separate base from aspects. There is no core system, and only one type of concern development process exists to create designs that realize crosscutting and non-crosscutting concerns [16]. Requirements are decomposed in a uniform fashion, which makes it possible to project any particular set of requirements on a range of other requirements [23], hence supporting a multi-dimensional separation of concerns [34].

In an asymmetric approach, parallel concern development processes coexist: one that creates and expands the base model and another that creates aspects that crosscut the base. An early identification of requirements aspects improves the base structure’s modularity since scattered and tangled stakeholder interests are clarified. A requirements aspect does not exist in isolation. It needs to be progressive, as suggested by the Twin Peaks model in Fig. 2, and interact with base requirements in order to provide some service according to its intent. This service providing process is specified by certain weaving mechanisms.

Table 1. Comparison and classification of asymmetric requirements aspects approaches

Base Form	Requirements Aspects	Advices	Join Points	Pointcuts	Weaving	Sample Approach
use case	extensions	extension behaviors	extension points	list of extension points	extension composition	[20]
goal model	softgoals	advising tasks	goals and tasks	contribution links	composing algorithm	[36]
multi-stakeholder description	terminological interferences	vocabulary mappings	conflicts and correspondences	requirements discordances	reconciling concepts	[24,25]
viewpoint	concerns	stakeholder requirements	viewpoints	contribution table	resolving conflicts	[30,31]
problem frame	security	vulnerabilities	functional requirements	threat descriptions	composition process	[15]
natural language	non-functional terms	occurrences	requirements statements	indicator terms	information retrieval	[17]

It should be noted that, depending on how requirements aspects are modeled and traced in the software life cycle, one may not define a specific weaving mechanism for every requirements aspect. That is why weaving mechanism is an optional feature shown in Fig. 1. Nevertheless, requirements approaches that support early aspects analysis unexceptionally treat weaving as a crucial component.

We use AspectJ-like terminologies to compare and classify existing aspect-oriented requirements analysis approaches according to their base forms. To define the terms used in the comparison, we follow the metaphor that every requirements aspect acts as a service provider to some base modules.²

- **Advice** defines the body or the content of the service that a specific requirements aspect provides. It describes *what* the service is about.
- **Join points** are points in the base which a requirements aspect interacts with. They describe *where* the service is provided.
- **Pointcut** represents a set of join points. It describes the *situational patterns* of the service that an aspect provides.
- **Weaving** is the process of coordinating service providers (requirements aspects) and consumers (base requirements). It describes *when* and *how* the service takes place.

² The following interpretations of the concepts may depart from the traditional explanations developed from the aspect-oriented programming perspective, which can be found in [1], for example.

In addition, the mandatory feature “intent” of a requirements aspect describes *why* the service is needed in the first place. We interpret the above concepts from a requirements engineering perspective, as part of the primary contribution of this paper. Table 1 summarizes the comparison of requirements aspects approaches based on our proposed taxonomy and terminologies. We classify the asymmetric approaches according to their base forms, and provide sample references for each category. This examination and classification were performed independently by the first author and the third author following a pre-defined artifact analysis protocol [8], and the differences were reconciled.

Note that the comparison presented in Table 1 is not intended to be exhaustive or complete by any means. What we desire is a way of framing our surveyed representative approaches under the umbrella of our conceptual framework, so that Table 1 both exemplifies the taxonomy’s use and substantiates its value. One of our future research areas is to extend the comparison results in Table 1 to cover all the features presented in Fig. 1.

3.3 Kind

Aspects at the requirements level can be classified according to their kind, i.e., the matter they concern. Requirements standards and textbooks typically classify requirements into *functional* requirements on one hand, and *non-functional* requirements or attributes on the other hand. In this classification, requirements given in terms of required operations or data are considered to be functional, while quality requirements, such as performance, security, reliability, maintainability, are classified as non-functional.

A functional requirement can pertain to a function or to data. In mathematical terms, given any element, say x , in the domain, a function assigns exactly one element in the co-domain to x . Thus, a widespread function that a system shall perform, such as logging and tracking memory usage, is a functional requirements aspect. A functional aspect can also describe the data item or data structure that shall be part of a system’s state, e.g., data integrity for multicast overlay networks. Besides, functional aspects can address crosscutting architectural and technical concerns like buffering and caching, as the Twin Peaks model in Fig. 2 shows.

Non-functional requirements address system qualities. They are typically broad in scope in that they have impacts on multiple requirements modules. Sample non-functional aspects include: performance in terms of time (points in time, reaction time, time intervals), speed, volume, throughput, or rates (volume per time unit), and properties of product management, such as availability, testability, interoperability, portability, etc. Security – one of the most mentioned non-functional requirements aspects – will be further discussed in section 4.

Note that there exist many requirements classification frameworks other than the functional versus non-functional one. For example, the satisfaction of requirements can be hard or soft, the representation of requirements can be operational, quantitative, qualitative, or declarative, and so forth. Discussion of specific schemes to classify requirements into different kinds is beyond the scope

of this paper. What we want to emphasize here is that requirements aspects give rise to a separate dimension for stakeholders to concentrate on requirements crosscutting properties. And we have used the *de facto* classification scheme to show that both functional and non-functional requirements can be considered as aspects, provided that the concerns they address cut across other concerns of the base requirements structure.

3.4 Crosscutting Properties

In explaining and elaborating the proposed taxonomy, we use the feature diagram shown in Fig.1 as a baseline to present diverse characteristics a requirements aspect must or can have. As we choose features to organize our discussion, “aspects” that crosscut more than one feature emerge. We now discuss some of these crosscutting properties as follows:

- *Constraints*. A constraint is a limitation of possibilities. For example, a common constraint to the identification and modularization of aspects is the amount of time available for early requirements engineering activities.
- *Dependencies*. Determining a particular feature may rely on other features. For example, weaving mechanism highly depends on requirements base forms and related aspect concepts, as manifested in Table 1.
- *Trade-offs*. Trade-offs involve determining the optimum balance among certain factors. A trade-off analysis is a systematic examination of the advantages and disadvantages of the alternatives. For example, choosing between mapping and influence as target for an identified requirements aspect in the software life cycle invokes detailed trade-off analyses.
- *Rationale*. A rationale records an underlying reason or an explanation of controlling principles of opinion, choice, or practice. For example, the rationale of using natural language as the base form is to facilitate the contractual process for requirements procurement.
- *Consistency*. Maintaining consistency helps achieve harmony of features to one another and as a whole. For example, resolving conflicts is crucial for viewpoint-based frameworks to weave requirements aspects, as indicated in Table 1.

The above is only a partial list of interplays between features shown in the taxonomy. Nevertheless, it illustrates the necessity and importance of understanding these interplays for effectively managing requirements aspects.

4 A Reified Requirements Aspect – Security

To further explore the usefulness of the taxonomy, we study one of the most discussed requirements aspects in the literature – security, which tends to be stated as a crosscutting concern that impacts many requirement-level concerns and artifacts.

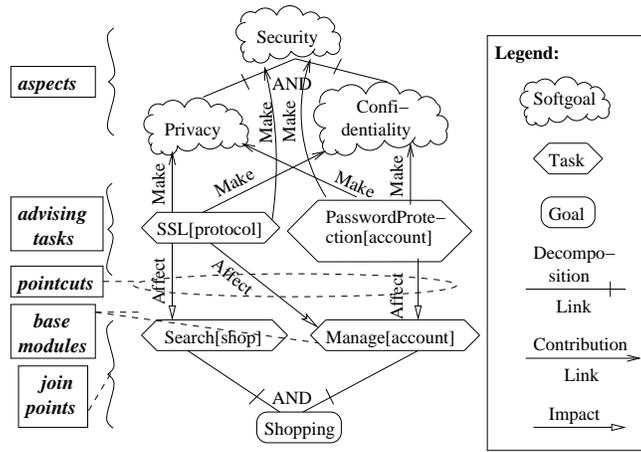


Fig. 3. Security aspects in requirements goal models (Adpated from [25]).

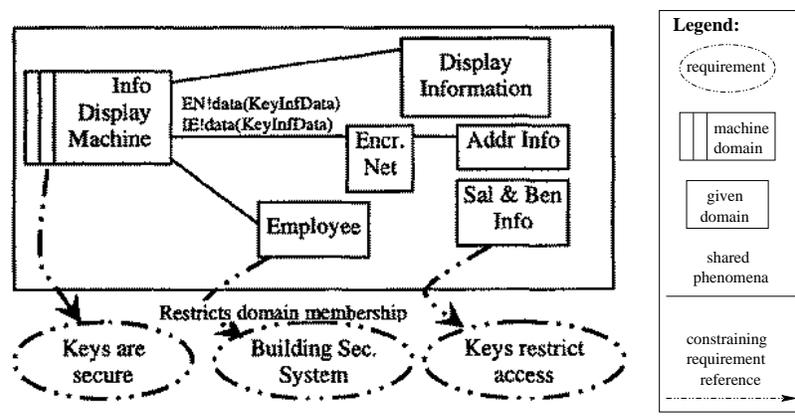


Fig. 4. Security aspects in problem frames (Adpated from [15]).

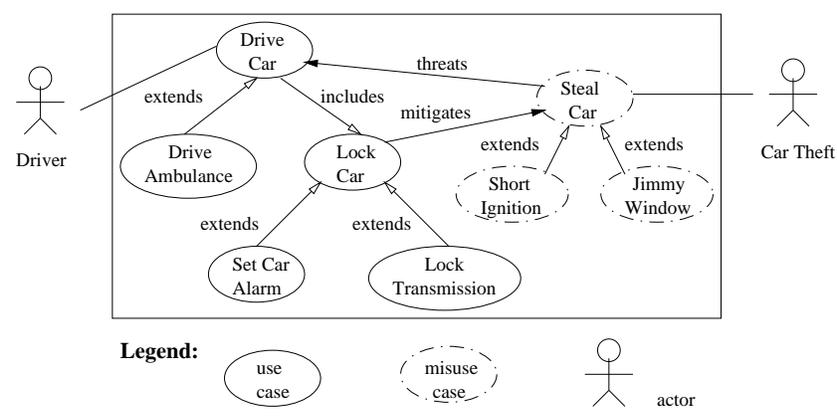


Fig. 5. Security aspects in (mis)use cases.

Security requirements are concerned with how assets are to be protected from harm [22]. An asset is something in the context of the system, tangible or not, that is to be protected [18]. A threat is the potential for abuse of an asset that will cause harm in the context of the problem. A vulnerability is a weakness in the system that an attack exploits. Security requirements are constraints on functional requirements intended to reduce the scope of vulnerabilities. Thus, security requirements stipulate the elimination of vulnerabilities that an attacker can exploit to carry out threats on assets, thereby causing harm [15].

In many practices, the security needs of a given system are often not determined until well into the implementation, resulting in late and expensive attempts to shoehorn security into the work in progress [15]. This leads to the situation that security requirements are often stated in terms of *how* (e.g., the system shall use cryptography), and not in terms of *what* problems to be solved in the first place, leaving it unclear how the security requirements affect other stakeholder concerns.

Treating security as a requirements aspect helps first determine what crosscuts what, and then tease out the implications of this crosscutting. We demonstrate the applicability and competence of the taxonomy described in section 3 by delineating sample aspect approaches that deal with security requirements. The discussions are structured by the base forms of the authors' chosen organizations: goal models, problem frames, and use cases. A more detailed investigation that aims to cover all features in Fig. 1 is currently under planning.

The context of Fig. 3 are goal models developed for a media shop to investigate requirements for new on-line services [25]. In Fig. 3, security is expressed as a softgoal to suggest that intended software is expected to satisfy the security requirements within acceptable limits, rather than absolutely [7]. As Table 1 indicates, softgoals in a goal model are requirements aspects, and concrete tasks such as SSL[protocol] are advices which are to be weaved into proper join points (hard goals and functional tasks). Composing algorithms are defined in [36] to guide the weaving procedure.

Figure 4 illustrates the work on deriving security requirements from crosscutting threat descriptions [15]. Problem frames are used as requirements base form to provide a shape of a solution for various problem classes [19]. In Fig. 4, functional requirements, such as "salary and benefits information shall be available to each managerial employee for perusal", are represented by problem frames and give vulnerabilities. Threat descriptions like "releasing salary information to unauthorized individuals could damage the company's reputation" help determine which functional requirements are exposed to specific vulnerabilities. Vulnerabilities are then ameliorated by security requirements, which are expressed as frames' constraints like "keys restrict access" and "restricts domain membership". Details of the composition process of weaving threat descriptions into problem frames are discussed in [15].

A misuse case is a use case from the point of view of an actor hostile to the system under design. Its goal is not a system function, but a threat posed by that hostile actor. One of the applications of misuse cases is to elicit security require-

ments [2]. Figure 5 shows a scenario in which malign agents and misuse cases are explicitly modeled. This helps to focus attention on the struggle against exposed threats. For example, in Fig. 5, the car theft’s intention is to break into the car and to short the ignition, thus stealing the car, possible mitigating approaches are to set the car alarm, or lock the transmission. According to Jacobson [20], extensions in UML share much of the spirit of aspects in aspect-oriented software development: they both allow developers to slice a system over all life cycle models. It is important to keep the extension mechanism nonintrusive, i.e., aspects extend the base use case with more behavior without having to change the base. The goal is to achieve modular design and code by structuring them from a base and let the system grow without cluttering the base with statements that have nothing to do with the base, even if the statements are important for the additional behavior [20].

Discussion of the above approaches shows that our proposed taxonomy of requirements aspects indeed serves as an umbrella for a variety of ideas. The non-functional security requirements intend to protect assets from harm. They cut across many author’s chosen base structures: goal models, problem frames, and use cases. They are modularly represented, successively refined, and systematically composed with functional requirements in order to guide development throughout the entire software life cycle.

5 Related Work

The early aspects Web portal [12] remains one of the most comprehensive resources for investigating current challenges and exploring future directions on aspect-oriented requirements engineering and architecture design. Two early aspects landscape reports are available in the Web portal.

In [4], the authors lay out a conceptual landscape for how the concept of early aspects pertains to domain analysis, requirements engineering, and architectural design. The report distills all the concepts special to each, and provides a conceptually unified view of early aspects in general. An extensive survey of aspect-oriented requirements engineering approaches is provided. An aspect at the requirements level is defined as a broadly-scoped property, represented by a single requirement or a coherent set of requirements (e.g., security requirements), that affects multiple other requirements in the system so that: 1). it may constrain the specified behavior of the affected requirements; 2). it may influence that affected requirements in order to alter their specified behavior [4]. Our taxonomy of requirements aspects enhances this view with emphasis on the intent behind this constraining and influence. Besides the behavioral view, our proposed taxonomy also captures requirements aspects’ structural and life cycle properties.

The authors of [6] provide a survey of contemporary aspect-oriented analysis and design approaches, considering their origins, aims, and contributions. A very general description is given for requirements aspects: “if some requirements are not effectively modularized, . . . such requirements are termed crosscutting (or as-

pectual) requirements” [6]. In order to compare existing aspect-oriented analysis and design approaches, the authors define some metrics: traceability, composability, evolvability, and scalability. While our taxonomy clearly fine-tunes the description about requirements aspects provided in this survey, the comparison criteria characterized in [6] supplement our work in quality issues of requirements aspects approaches.

Requirements aspects often recur in the literature. For instance, security is the most prevalent requirements aspect discussed in nearly all aspect-oriented requirements engineering approaches. Orthogonal to the taxonomy proposed in this paper, catalogue or ontology-based approaches attempt to collate, from a wide range of sources, verifiable information in specific domains. Along this line, the most relevant work to requirements aspects is the non-functional requirements catalogue devised by Chung and his colleagues [7]. Non-functional requirements are hard to be allocated into independent modules, therefore have huge potential to become candidate early aspects [25]. One of the main motivations of catalogue or ontology building is the possibility of knowledge sharing and reuse: as soon as a particular domain is fixed (e.g., security of a media shop as shown in Fig. 3), it seems reasonable to expect a large part of domain knowledge to be the same for a variety of applications (e.g., security of information display in Fig. 4 and of car antitheft in Fig. 5), so that the high costs of knowledge acquisition can be better justified. Non-functional requirements or requirements aspects catalogue building could benefit from our work since the taxonomy helps filter out feature nuggets one needs to consider when cataloguing requirements aspects.

The idea of using feature diagrams to depict a concept has become popular in software engineering, since feature modeling overcomes the drawback of presenting domain knowledge in a flat and coarse form. Recent work has been carried out in understanding both familiar and relatively new concepts. For example, the notion of components is investigated by Voelter [35], and a feature-based survey of model transformation approaches is provided by Czarnecki and Helsen [10]. These pieces of work greatly inspired us to explore the notion of requirements aspects. The taxonomy proposed in this paper is, to the best of our knowledge, the first attempt to characterize the concept of requirements aspects using feature diagrams in order to efficaciously serve the early aspects community.

6 Summary and Future Directions

Maintaining a clear separation of concerns throughout the software life cycle has long been a goal of the software community. Aspects provide the mechanism that enables the source code to be structured to facilitate the representation of multiple perceptions and to alleviate tangling and scattering concerns. Many of these concerns often arise in the problem domain [28], and, therefore, it is important to identify and represent concerns that arise during the early phases of software development, and to determine how these concerns interact. The Twin Peaks model [27] suggests that at the stage when requirements need to be

mapped onto elements of a software solution, identifying aspects may become much more worthwhile.

In this paper, we have presented a preliminary attempt to structure the requirements aspects domain with a taxonomy, characterizing mandatory, optional, and alternative features of a requirements aspect as it is used today in the literature. We then use the taxonomy to compare and classify existing requirements engineering approaches that handle aspects. We have also studied crosscutting security requirements to exemplify the taxonomy's use, substantiate its value, and demonstrate its applicability and usefulness.

The work reported here can be continued in many directions. Obviously, the literature study we based the taxonomy could be expanded, either within the published work or by consulting researchers and practitioners in the community. Also of interest would be extending the taxonomy to cope with symmetric, multi-dimensional concern separation, approaches. Moreover, the future research agenda includes investigating the taxonomy's applicability to handle architectural-, code-, and test-level aspects. Furthermore, it is necessary to establish comparison criteria and metrics in order to conduct a more critical examination that is intended to clarify what is better and/or worse in every considered approach. Finally, experience from practical applications will have to be evaluated with respect to the taxonomy's usefulness: should it be modified, simplified, or extended with additional features?

The area of early aspects continues to be a subject of intense research. The time is ripe for consolidating early aspects knowledge into a set of polished best practices and patterns. We hope the proposed generic, feature-based taxonomy of requirements aspects can be an important contribution to existing knowledge, advance the current state of the art, and serve as a basis for more rigorous investigations in this area.

Acknowledgments. *We would like to thank Awais Rashid for helpful remarks, and Krzysztof Czarnecki for help with feature modeling. Financial support was provided by NSERC.*

References

1. Aspect-oriented software development community wiki: <http://www.aosd.net/wiki/>
Last accessed on April 16, 2007.
2. I. Alexander. Initial industrial experience of misuse cases in trade-off analysis. In *Intl. RE Conf.*, pages 61–68, 2002.
3. J. Araújo, J. Whittle, and D-K. Kim. Modeling and composing scenario-based requirements with aspects. In *Intl. RE Conf.*, pages 58–67, 2004.
4. J. Araújo, E. Baniassad, P. C. Clements, A. Moreira, A. Rashid, and B. Tekinerdoğan. Early aspects: the current landscape. Technical Report COMP-001-2005, Lancaster Univ., 2005.
5. E. Baniassad, P. C. Clements, J. Araújo, A. Moreira, A. Rashid, and B. Tekinerdoğan. Discovering early aspects. *IEEE Software*, 23(1):61–70, 2006.
6. R. Chitchyan, A. Rashid, P. Sawyer, A. Garcia, M. Alarcon, J. Bakker, B. Tekinerdoğan, S. Clarke, and A. Jackson. Survey of aspect-oriented analysis and design approaches. AOSD-Europe-ULANC-9, AOSD Europe, 2005.

7. L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.
8. Course Website – Empirical Research Methods in Software Engineering: <http://www.cs.toronto.edu/~sme/CSC2130> Last accessed on April 16, 2007.
9. K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.
10. K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3):621–645, 2006.
11. R. P.-Díaz. Domain analysis: an introduction. *ACM SIGSOFT Softw. Eng. Notes*, 15(2):47–54, 1990.
12. Early aspects portal: <http://www.early-aspects.net/> Last accessed on April 16, 2007.
13. O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *Intl. Conf. on RE*, pages 94–101, 1994.
14. J. Grundy. Aspect-oriented requirements engineering for component-based software systems. In *Intl. Symp. on RE*, pages 84–91, 1999.
15. C. B. Haley, R. C. Laney, and B. Nuseibeh. Deriving security requirements from crosscutting threat descriptions. In *Intl. Conf. on AOSD*, pages 112–121, 2004.
16. W. H. Harrison, H. L. Ossher, and P. L. Tarr. Asymmetrically vs. symmetrically organized paradigms for software composition. RC22685, IBM Thomas J. Watson Research Center, 2002.
17. J. Cleland-Huang and R. Settini and X. Zou and P. Solc. The detection and classification of non-functional requirements with application to early aspects. In *Intl. RE Conf.*, pages 39–48, 2006.
18. ISO/ICE: information technology – security techniques – evaluation criteria for IT security. Geneva Switzerland: ISO/IEC, 1999.
19. M. Jackson. *Problem Frames*. Addison Wesley, 2001.
20. I. Jacobson. Use cases and aspects – working seamlessly together. *Journal of Object Technology*, 2(4):7–28, 2003.
21. K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon Univ., 1990.
22. J. D. Moffett and B. Nuseibeh. A framework for security requirements engineering. YCS368, Dept. of Computer Science, Univ. of York, 2003.
23. A. Moreira, J. Araújo, and A. Rashid. A concern-oriented requirements engineering model. *LNCS*, 3520:293–308, 2005.
24. N. Niu and S. Easterbrook. Analysis of early aspects in requirements goal models: a concept-driven approach. *Trans. on AOSD*, 2007 (to appear).
25. N. Niu and S. Easterbrook. Discovering aspects in requirements with repertory grid. In *Early Aspects Wkshp at ICSE*, pages 35–41, 2006.
26. B. Nuseibeh and S. M. Easterbrook. Requirements Engineering: A Roadmap. In *The Future of Software Engineering*. IEEE Computer Society Press. 2000.
27. B. Nuseibeh. Weaving together requirements and architectures. *IEEE Computer*, 34(3):115–117, 2001.
28. B. Nuseibeh. Crosscutting requirements. In *Intl. Conf. on AOSD*, pages 3–4, 2004.
29. B. Ramesh and M. Jarke. Toward reference models for requirements traceability. *IEEE Trans. Softw. Eng.*, 27(1):58–93, 2001.
30. A. Rashid, P. Sawyer, A. Moreira, and J. Araújo. Early aspects: a model for aspect-oriented requirements engineering. In *Intl. RE Conf.*, pages 199–202, 2002.
31. A. Rashid, A. Moreira, and J. Araújo. Modularisation and composition of aspectual requirements. In *Intl. Conf. on AOSD*, pages 11–20, 2003.

32. A. Rashid, and A. Moreira. Domain models are not aspect free. In *Intl. Conf. on MoDELS/UML*, pages 155–169, 2006.
33. P-Y. Schobbens, P. Heymans, and J-C. Trigaux. Feature diagrams: a survey and a formal semantics. In *Intl. RE Conf.*, pages 139–148, 2006.
34. P.L. Tarr, H. Ossher, W. H. Harrison, and S. M. Sutton. N degrees of separation: multi-dimensional separation of concerns. In *ICSE*, pages 107–119, 1999.
35. M. Voelter. A taxonomy of components. *Journal of Object Technology*, 2(4):119–125, 2003.
36. Y. Yu, J. C. S. do Prado Leite, and J. Mylopoulos. From goals to aspects: discovering aspects from requirements goal models. In *Intl. RE Conf.*, pages 38–47, 2004.