|  |  |
| --- | --- |
| Código | Segmento |
| Definición de interés | Concerns are pervasive in software development.  They originate in the interest s of the stakeholders of a software system, including customers, developers, users. They arise throughout the software life cycle, from ideas for features or functions that inspire new development, through to properties such as maintain-ability and adaptability that affect the utility and value of a product long after initial development. |
| Definición de interés | A concern is any domain of interest to a software developer. A concern has a root phase, where the concern manifests itself for the first time, and encapsulates a set of models. |
| Definición de interés | The IEEE defines the “concerns” for a system as “… those interests which a pertain to the systems development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders.” Concerns arise throughout the software life cycle, and individual concerns commonly span multiple life-cycle stages and work products. |
| Definición de interés | Commonly, the term concern is overloaded, in respect to programming structures, to be anything of interest of a software system. Thus, we restrict a concern to a logical classification of a source code fragment. Programming languages support, by varying terms, the application of SOC |
| Definición de interés | As defined in [], a concern is a modular unit of reuse that encapsules a set of models describing all properties of a domain of interest during software development, often spanning multiple phases of software development and levels of abstraction (from requirements and analysis models to design models to code) |
| Definición de interés | Each concern has a root phase where the concern manifests itself for the first time. Some concerns appear in early phases of software development, e.g., broadly scoped system properties with functional, non-functional, or even intentional characteristics. Some concerns appear in later phases of software development, e.g., solution-specific concerns such as specific communication protocols and design patterns. |
| Definición de interés | At this point, a concern in FEAT is simply a named container for a fragment of a program which is of interest to a user. Any class, method, or field in a project can then be moved to the FEAT perspective. |
| Definición de interés | To promote concerns to first-class entities (“concerns”) in software development, they must he defined indepently of any specific type of software artifact and even of software artifacts in general. One dictionary definition od concern is “a matter for consideration” |
| Definición de interés | More specifically to software, the IEEE defines the concerns for a system as "... those interests which pertain to the system's development, its operation or any other aspects that are critical or otherwise important to one or more stakeholders" [I0, p. 4] |
| Importancia de los intereses | Although, the key motivation is to analyze the architecture with respect to the concerns it appears that concerns are not explicitly represented as first class abstractions in scenario-based analysis approaches. This is somehow surprising since the primary motivation for analyzing the architect ure is in fact the analysis of the stakeholder concerns. In general a concern is defined implicitly in the scenarios and the evaluation of the architecture is performed for scenarios. |
| Importancia de los intereses | If you examine software development projects, there are several recurring scenarios that cause problems to the project, often causing them to fail altogether. |
| Importancia de los intereses | Concerns are what we care about in software. Software is developed to address concerns expressed in functional and nonfunctional requirements. Software must further reflect and accommodate concerns that arise in the development process |
| Importancia de la entidad de primera clase | Disentangle latent concerns or aspects from existing software, making them first-class entities. |
| Importancia de la entidad de primera clase | In order to explicitly reason about traceability of the concerns in architectural views it is necessary that the corresponding concerns are explicitly modeled as first class abstractions |
| Ventajas de la separación de intereses | Advantages of separation of concerns include the ability to partition development efforts according to concerns and the ability to isolate changes, minimizing modification efforts and limiting the impact on unrelated parts of a system. |
| Ventajas de la separación de intereses | Effective separation of concerns should make software easier to maintain, evolve, customize, adapt, adopt, integrate, and reuse. |
| Deficiencias de la separación de intereses | The representation, separation, and integration of concerns are important concepts in software development. Nevertheless, many problems with software can still be traced to the limitati ons of current approaches to managing concerns. These include the a priori imposition of particular concerns or types of concern, the failure to treat concerns as first-class entities across the software life cycle, and the imposition of rigid decompositions on software artifacts. |
| Deficiencias de la separación de intereses | Separation of concerns is a well-established principle of software engineering. Nevertheless, the failure to separate concerns effectively has been identified as a continuing cause of the ongoing software crisis. This failure arises in part because most programming and modeling formalisms enforce a dominant decomposition that allows only a few concerns to be separated, whereas software in reality is subject to multiple simultaneous, overlapping, and crosscutting concerns. |
| Deficiencias de la separación de intereses | Although ASOC has been emphasized in recent work, concerns themselves have remained something of second-class citizens. Current ASOC tools provide only limited support for explicit concern modeling, representations of concerns tend to be tied to particular tools or artifacts, and concern modeling usually occurs just in the context of a particular type of development activity. such as coding or design [4, 14, 25]. A global perspective on concerns, that spans the life cycle and is independent of particular development tools or artifacts, has been lacking |
| Deficiencias de la separación de intereses | In reality, often modules are responsible for multiple concerns. Furthermore, concerns are scattered over various modules [17]. Consequently, programmers face the challenge to comprehend the concerns and their encoding. |
| Deficiencias de la separación de intereses | All approaches that intent to describe and visualize a complex scenario share some shortcomings: Each visualization is either easily understandable and wrong or hard to understand and no less wrong. Each model leaves out the one vital element you need unless the model is as complex as reality, at which point it becomes unusable. |
| Deficiencias de la separación de intereses | This is a very common effect [4] in software projects. A ripple-effect occurs when in a chain of concerns C1 – C2 – C3 – C4 a minor change to C1 requires a minor adjustment in C2, which in turn requires a change to C3, which again requires a change to C4. Thus, minor changes to a single concern could (and often does) affect a large number of other concerns (and the concerns these are related to). Ripples are exceedingly hard to identify beforehand because the concerns involved in the change are spread across several brains and areas of responsibility. In complex concern models, the ripple effect is as common as in the projects they represent and even here, it's impossible to identify them based on the concern model alone. Once a possible ripple is found, though, it is possible to determine its impact on the project as a whole |
| Deficiencias de la separación de intereses | Unfortunately, despite the ongoing efforts for identification and modeling of concerns in architectural views, the traceability of concerns rema ins a challenging issue in architecture design. In the aspect-oriented software development community the interest is in particular on crosscutting concerns which cannot be easily localized and are scattered over multiple implementation units. Several approaches have already been proposed to model crosscutting concerns at the architecture design level [6][2] |
| Deficiencias de la separación de intereses | Albeit the primary motivation is to analyze the impact of stakeholders’ concerns, it appears that concerns are not explicitly represented as first class abstractions. The lack of an explicit notion of concern in scenario-based analysis approaches can result in an incomplete analysis because scenarios are too specific and can only partially represent the concerns |
| Deficiencias de la separación de intereses | Current scenario-based software architecture analysis methods aim to analyze the impact of stakeholder concerns on the architecture. Unfortunately the notion of concern is not a first class abstraction in these approaches and the analysis is primarily based on the impact of scenarios. This leads to a partial understanding of the impact of concerns and as such provides risks for the optimal refactoring of the architecture. |
| Deficiencias de la separación de intereses | Object-oriented languages, for example, support decomposition (with abstraction, encapsulation, and modularity) according to a particular hierarchy of object classes. However, this does not allow for decomposition (especially concurrent decomposition) along other dimensions such as features or functions, and it allows only a particular decomposition of object classes to be expressed. This sort of limitation has been characterized as the “tyranny of the dominant decomposition” [19]. This is problematic for system evolution because software is subject to many concerns. At any given time, a system is subject to. |
| Deficiencias de la separación de intereses | As argued in [8], this is due at least in part to an imbalance between the complexity of concerns and the capabilities of current mechanisms for separating concerns. Any given software component is subject to multiple, simultaneous, overlapping concerns, whereas most programming languages and other development formalisms support the systematic separation of only a small number of concerns (such as classes or functions). As a result, many concerns must crosscut the primary decomposition and become entangled with it. Changes related to crosscutting concerns are consequently not isolated and may entail disproportionately large impacts and costs. |
| Tipos de intereses | Abstract concerns include conceptual entities, such as features, properties, topics of interest, and so on. Concrete concerns represent physical entities, notably the work products of software development |
| Tipos de intereses | Concerns [9] seem to correctly capture the intent of this approach: they encapsulate elements of different types that are of interest in a particular context. There are different types of concerns ranging from application-specific such as features to user-oriented such as performance. More explicitly, we see the exploration findings as the expression of a concern and the basis for a process or a guide. |
| Tipos de intereses | Cosmos divides concerns into two categories, logical and physical. Logical concerns represent concerns viewed conceptually, as “matters for consideration”: issues, problems, “ilities”, and so on. Logical concerns are concerns as we usually think of them. Physical concerns comprise the actual things that constitute our systems, such as specific work products, software units, hardware units, and services. Physical concerns are a way to bring the “real world”, to which our (logical) concerns apply, into the concern-modeling space. Logical concerns are further typed as kinds, instances, properties, and topics. Kinds are categories of concerns (such as functionality, behavior, and state). Instances are particular concerns (usually of some kind, e.g., particular functions, behaviors, states). Properties are characteristics of kinds and instances. Topics are (typically theme-related) groups of concerns of generally different types. Physical concerns comprise instances, collections, and attributes. Physical instances represent particular system elements (such as source files, design documents, workstations). Collections represent groups of these. Attributes are the specific properties of instances or collections, such as the size of a design document or number of files in a directory. |
| Relaciones entre intereses | Concerns do not occur in isolation, so it is necessary to represent relationships among them. These arise from the semantic properties of artifacts, dependencies in the development process, stakeholder viewpoints. Relationships among concerns contribute to an understanding of software and help to organize development tasks. For broad applicability, we must be able to represent relationships of arbitrary kinds, among arbitrary types of elements . Moreover, relationships should be first-class in the concern model. |
| Relaciones entre intereses | Relationships i n C onMan represent rel ationships among concern model elements. Relationships are typed in the ConMan schem a by t heir st ructure and access methods. They can be binary or n-ary, and the elements in them can be ordered or not . In implementation, ConMan relationships are not typed by relationship semantics; so, for exam ple, di rected bi nary rel ationships are used t o represent many kinds of sem an-tics, such as implements, extends, refersTo, and others among Java artifacts. |
| Relaciones entre intereses | The Relationship view shows the relationships between the active concern and other concerns. For example, if the user is currently following a guide, this view will show the existence of a relationship between the guide and the result being produced. The user can navigate among concerns by double-clicking on a concern. This activates the new concern and refreshes the relationship view to present the new relationships. |
| Relaciones entre intereses | To distinguish from the previous intra-view traceability we use the term inter referring to traceability relations across different views. In principle, there are two kinds of relations. First, architectural elements in different views might be related, this is called, inter element to element traceability. Second, a common concern might be related to architectural elements in different views, which is termed as inter concern to element traceability. |
| Relaciones entre intereses | Relationships are divided into four categories: categorical, interpretive, physical, and mapping. Categorical relationships reflect fundamental semantics of the concern categories. These relationships include kind-of, which relates (sub)kinds to kinds, instance-of, which relates instances (both logical and physical) and properties to kinds, applies-to, which relates properties to kinds and instances, part-of, which relates instances to instances and properties to properties, member-of, which relates physical concerns (instances or collections) to collections, and relates-to, which relates any concern to a topic. |
| Relaciones entre intereses | Interpretive relationships relate logical concerns with semantics that are not based on categories. One example is contributes-to, which indicates that one concern (e.g., logging behavior) contributes in some way to another (e.g., robustness). Another interpretive relationship is motivates, which indicates that one concern (e.g., robustness) motivates another (e.g., logging). Such relationships are especially important in understanding the system-dependent semantics of concerns and are typically defined with respect to particular concern-modeling objectives. Physical relationships represent associations among physical concerns (e.g., composition relationships among Java classes in Hyper/J [6]). Mapping relationships represent non-categorical associations between logical and physical concerns, for example, the implementation of a logical function by a Java class. These are of special importance in modeling concerns for purposes of composing component implementations. Predicates represent integrity conditions over various relationships and can be classified accordingly. |
| Modelado de intereses | ConMan concern s represent the conventional notion of a concern and can contain other concern model elements. Concern contexts (which specialize concern) can also have associated relationships and constraints. The structure of a concern space can be built up by the grouping of model elements and by explicit relationships and constraints. |
| Modelado de intereses | Concerns can be assigned elements extensionally, intensionally, or by a combination of these. Extensional concerns (e.g., “Printing”) are assigned their elements directly. Intensional concerns (e.g., “Naming”) obtain their elements through evaluation of an associated query. A particular subtype of concern context is composition, which contains a set of elements to be composed and a set of composition relationships. |
| Modelado de intereses | Concerns, relationships, and constraints are all specializations of concern model element , the basic type of ConMan models. As such, al l have fi rst-class status, and al l can be appl ied to one another. Consequently, any concern model element can be loaded into a concern space, grouped i nto a concern, related by a relationship, or const rained by a constraint. These elements also support m ultidimensional separation of concerns (MDSOC) and crosscutting associations. For example, in Figure 1 the feature “Naming” occurs under both the “Features” concern and al so under a “Topic” co ncern to wh ich it is related. Units (discussed below) can also cro sscut m ultiple concerns, relationships, and constraints in a concern space, such as the interface org.eclipse.cme.Entity in Figure 1. |
| Objetivo del modelado de intereses | Identify, model and visualize concerns and aspects in software and relationships among concerns and software units (i.e., pieces of software artifacts of any type, including both code and non-code artifacts). |
| Objetivo del modelado de intereses | Concerns can be identified upfront, during the software’s initial creation, or on demand, at any point during the software’s lifecycle when the concern manifests itself. |
| Patrón para identificar intereses | As a result of this scenario, you get some code that is ill thought-out, badly integrated and hastily implemented. The concern model of your application often directly reflects this as missing artifacts. The missing artifacts make traceability of any the requirement itself very hard. Also, any form of integration, impact analysis and evolution of the code is extremely difficult as there is no clear understanding of what other components are affected by the code injected into the application. Definition: This pattern describes two concerns C1 and C2 which are connected by a relationship R1 and not within adjacent development phases. |
| Patrón para identificar intereses | It should be noted that this pattern is hard to find since often, the developers will not admit (or are even aware) that the documents are not well formed and contain information that should be part of a different phase. At the moment, manual intervention (reading the artifacts in question) is often needed to correctly identify this pattern. Definition: This pattern is the representation of the Dependency Inversion Principle.[3] It shows a concern C2 to reference a concern C1 where C1 is nominally in an earlier phase of the development. |
| Patrón para identificar intereses | This scenario is basically a user error as it should not occur at all. The simple rule „one method, one task“ should be applicable to higher-level artifacts as well, but in most cases, it is not applied. As a result from this pattern, all concerns downstream from C3 are no longer clearly traceable from C1 or C2 as they are not uniquely attributable to either concern. Definition: This pattern is similar to the God-class phenomenon found in code. Multiple concerns converge in one concern at a later phase of the development. The concerns C1 and C2 converge in C3. The relationships R are of different, nonuniform types. |
| Patrón para identificar intereses | Analysis: This pattern indicates an over-separation of concerns on one of the earlier phases or an under-separation of concerns on the implementation level. Definition: Similar to the Convergence Pattern, but R1 and R2 are logically implements relationships and R3 and R4 are physically implements relationships. |
| Patrón para identificar intereses | Definition: This pattern shows that C2 and C3 are sub concerns of C1 that are refined independently into C4 and C5, are again treated as one concern C6 in a later phase. |
| Patrón para identificar intereses | Definition: This pattern features two concerns C1 and C2 in one phase and a third concern C3 in the next phase. As both C1 and C2 reference C3, any change to C3 must be reflected in C1 and C2 but since C1 and C2 have a relationship as well, any chance to either C1 or C2 needs to be checked against the other concern, which again can cause a change in C3. |
| Modelado de intereses en la implementación | Individual software units are seen to address or embody multiple concerns simultaneously, and these concerns can be organized in multiple, overlapping dimensions. It is this “hyperspaces” view of concerns that has motivated Cosmos. Hyper/J works on Java class files and supports both the decomposition and composition of Java programs. We focus here on composition. A composition is specified in three parts. The first defines the concern space, that is, the classes and packages that are available for composition. The second defines the concern mapping, that is, which classes are assigned to which concerns in which dimensions. The third defines a hypermodule, that is, a composed module comprising units from multiple concerns. The hypermodule specification has two parts. The first is a listing of dimensions and concerns that are involved in the composition. The second is a specification of composition rules indicating how various units assigned to the listed concerns are to be combined. The rules operate on units that include classes, interfaces, operations (abstract methods), actions (concrete methods), and fields. |
| Modelado de intereses en la implementación | Elements in the FEAT Perspective can be analyzed for their dependencies to other elements in the source code. Any element or relations of interest to a user can be stored as part of a concern. The source code corresponding to any element or relation can also be viewed in code viewer (bottom window). Concerns descriptions can be used to systematically analyze the code for a concern, or to compare two different concern descriptions. In both cases, FEAT automatically detects relations between elements in concerns, and presents the relations visually to the users. The architecture of FEAT consists in three components: a model component providing operation on concerns and ensuring the consistency of the concern description, an Java bytecode analyzer component, built on top of IBM's Jikes Bytecode Toolkit [4], which provides relations between different elements in a program, and a GUI component which ties the tool into the Eclipse Platform. |
| Restricciones en la herramienta | ConMan does not yet have a const raint definition language, but Java can be used to provide an implementation of constraint semantics, as can CME queries. |
| Restricciones en la herramienta | ConMan is a generalization of Hyperspaces as first proposed in [15] and implemented in Hyper/J [9]. Hyper/J used a multidimensional concern model but one that was flat and lacked support for general relationships and constraints. |
| Restricciones en la herramienta | Our toolset is tightly integrated with Eclipse, especially the Java Development Toolkit |
| Categorías de intereses en la herramienta | Kinds of Model Increments in CORE CORE supports two kinds of model increments: feature-driven increments and reuse-driven increment. |
| Categorías de intereses en la herramienta | Reuse-Driven Increments. Fig. 1 (b) illustrates reuse-driven increments in the context of CORE. An application/concern is typically built by reusing many concerns. During the reuse process, a user-tailored version of the reused concern is produced, and the partial elements in its customization interface are concretized in the application. The incremented model resulting from this reuse process is at a diferent level of abstraction or domain. |
| Enfoque de la herramienta | Towards the first goal, the CME initially surfaces tool s upport for two major AOSD approaches: the next generation of Hyper/J and multidimensional separation of concerns [2], and AspectJ [1], while providing underlying support for a broad spectrum of AOSD approaches. |
| Modelado de intereses en la arquitectura | COSAAM consists of the three phases: preparation, analysis and transformation. The preparation phase establishes the artifacts used in the COSAAM evaluation: a candidate software architecture design and a collecti on of concerns of stakeholders. The analysis phase involves a ch aracterization a nd measurement of scattering and tangling of concerns and modules. The information provided during this analysis is used in the transformation phase, in which the candidate software architecture is transformed. An iteration of COSAAM consists of all the activities of the analysis and transformation phases |
| Modelado de intereses en la arquitectura | 2.1 Preparation Phase. In the preparation the basic inputs for the analysis are defined: a candidate software architecture design and a collection of concerns from stakeholders. The phase consists of two parallel activities: describe candidate architecture and concern identification |
| Modelado de intereses en la arquitectura | To describe the architecture conventional software architecture modeling approaches are applied [1]. The architecture is then mapped to a DSM. Figure 1a shows for example the DSM for an example Window Management System architecture. The acronyms EM, PM, WM and SM represent EventManager, ProcessManager, WindowManager and ScreenManager, respectively. |
| Modelado de intereses en la arquitectura | 2.1.2 Concern Identification. In the concern identification step we derive the concerns that are important for the stakeholders. For this two steps can be followed. First, concerns can be reused from existing projects, requirements specifications or domai n models. Second, concerns can also be derived from scenarios developed by stakeholders. For the latter case we define a scenario-scenario DSM and derive concerns based on clustering of scenarios |
| Modelado de intereses en la arquitectura | 2.2.1 Initialization of Concern-Module DMM. The first step in the analysis phase is the mapping of concerns to modules. For this we apply the so-called Domain Mapping Matrix (DMM). In contrast to DSM’s that represent the mapping of elements in the same domain DMMs represent the mapping between elements from different domains [10]. In COSAAM we use a DMM to show the mapping fro m concerns to architectural elements |
| Modelado de intereses en la arquitectura | The toolset is composed of four views and an editor, partially shown in Figure 1. The Concern Explorer view is used to represent the different concerns present in a user workspace. A user can create different types of concerns in this view and interact with them by adding, removing or restructuring references to elements. From this view, a user can activate a concern to indicate that s/he intends to work primarily in the associated area of interest. S/he can still manipulate other concerns, but some operations are restricted to the active concern. |
| Modelado de intereses en la arquitectura | The References view shows information about the elements contained in a concern. The user can link this view with other views or simply drop or paste a concern into it to see its content. |
| Modelado de intereses en la arquitectura | Concerns in the system are rarely stable and need to evolve in accordance with the changing requirements. To cope with the evolution at the architecture design level it is necessary that the dependency links between the architectural concerns in the architectural views can be easily traced. |
| Modelado de intereses en la arquitectura | Module View of CCS. The module view represents the structuring of implementation units, or modules. The module view of |
| Modelado de intereses en la arquitectura | Component and Connector View of CCS. The Component and Connector (C&C) view represents the structuring of elements which have run-time behavior, which are usually components and connectors.  Deployment View of CCS. The deployment view represents the allocation of software elements to hardware nodes  The detail of concern model could range from just a description of its name to a full semantic model including attributes such as stakeholder, the domain of the concern, the date it was raised, the impact that it has, etc. |
| Modelado de intereses en la arquitectura | To support tracing these relations among architectural elements and concern explicit needs to made explicit. This can be achieved when dependency relations are recorded as trace links. For this, like concerns, traceability links should also be specified as first class abstractions in the adopted traceability model. |
| Modelado de intereses en la arquitectura | We define here two types of traceability: (1) intra concern to element traceability and (2) intra element to concern traceability . Note that Architectural Element can be either an architectural relation or architectural entity. In this way concerns can be both linked to architectural relations and architectural entities. Further, since architectural entities may be composed of other sub-entities a single concern can then be attached to a composition of architectural entities |
| Modelado de intereses en la arquitectura | Concern is defined for one or more Stakeholder. Unit Model represents the Units to which the concerns apply. A unit refers to an artif act in the software life cycle. Here we focus on the architecture design phase. Architecture Model is a subclass of Unit and consists of one or more Architecture View which consists of one or more Architectural Element. Architectural Element includes in the metamodel is in fact a representation of the actual architectural elements. To re fer to the actual elements Architectural Element includes the attributes reference and name. |
| Modelado de intereses en la arquitectura | Entities may have sub-elements that are represented by children relationship. Architectural Aspect represents a specification of an architectural aspect, wh ich is associated to one or more entities |
| Modelado de intereses en la arquitectura | DSMs can be used to analyze the properties of complex applications. In DSM-based architectural analysis in particular the coupling between the architectural models are depicted and an optimal decomposition is aimed by reducing the couplings using predefined matrix operations. COSAAM consists of three basic processes. In the preparation phase scenarios are elicited. Based on clustering mechanisms in DSMs we derive a number of concerns. In the analysis phase together with the architectural elements, the extracted concerns are represented in a so-called Domain Mapping |
| Modelado de intereses en la arquitectura | Matrix (DMM). Together with DMM we have defined a set of heuristic rules for analyzing the concerns. Finally, in the transformation phase the result of th e analysis is used to redefine the architecture. One of the key c ontributions of COSAAM is that it defines explicit heuristics for supporting the DM-based analysis. |
| Característica del método o herramienta | COSAAM is an iterative software architecture evaluation and transformation method that enhances scenario-based architecture analysis methods with DSM-ba sed analysis. In particular COSAAM builds on the earlier Soft ware Architecture Analysis Method (SAAM) [2] and Aspect ual Software Architecture Analysis Method (ASAAM) [9]. |
| Característica del método o herramienta | COSAAM utilizes a concern-oriented approach and includes an explic it and systematic transformation step. |
| Característica del método o herramienta | Cosmos [17] is a concern-space modeling schema that addresses this limitation. It allows concerns and the relationships among them to be modeled as first-class, independent entities that are separable from particular life cycles, work products, formalisms, and technologies. |
| Característica del método o herramienta | Hyper/J, which has been developed at the IBM T. J. Watson Research Center, is a tool for concern-based composition of Java programs [6].1 Hyper/J is based on the notion of multidimensional separation of concerns. |
| Característica del método o herramienta | CORE is a software reuse paradigm that advocates the concern as its main development artifact, and comes with a metamodel and reference implementation. |
| Modelado de intereses en el diseño | The CME, including ConMan, is implemented as a set of Eclipse [6] plug-ins that provide perspectives and views related to concern modeling. The Concern Explorer view (Figure 1) displays a ConMan concern model in a tree-structured form comparable to that shown in the Eclipse Package Explorer. Loaders build concern models based on Eclipse workspaces and (optionally) user concern schemas. |
| Modelado de intereses en el diseño | In the new version of TouchRAM, concerns have replaced aspects as the main unit of reuse, and the TouchRAM metamodel had to be updated in consequence. A speciﬁc design solution is still modelled with a RAM model, but all solutions related to a design concern are now grouped within a concern. A concern speciﬁes a variation interface, which consists of a set of features. Each feature is realized by one or several RAM models. |
| Modelado de intereses en el diseño | Building a design concern is a non-trivial, time consuming task. It requires a deep understanding of the nature of the design concern to be able to identify the different features, to model the common properties and differences of the concrete solutions, and to express the impact of the different variants on high-level goals. This can only be done by a domain expert, i.e., someone experienced who fully understands the nature of the concern and the tradeoffs involved in the different available options. TouchRAM now allows such a domain expert to create a new concern, name it, and then create a list of features that the concern offers. Each feature can be associated with one or several RAM models that describe its detailed design. |
| Modelado de intereses en el diseño | Cosmos [1] is a rich model for concerns and their relationships. It features a wide variety of different concern and relationship types that allow not only a structural but also a semantic analysis of the concerns in an application. |
| Modelado de intereses en el diseño | CosmosModel is an implementation of the model in Java. It features the full scale of concerns, relationships and constraints available in Cosmos. Since there are many different artifacts in an application, it quickly became obvious that I needed an automated process to identify possible problems and so have added an analysis engine used to search for concern patterns within the Cosmos which indicate latent or missing concerns, errors and omissions in the development and possible sources of problems. |
| Modelado de intereses en el diseño | With Cosmos we present a complementary approach to advanced separation of concerns that addresses concern modeling as an independent concern in software development. |
| Modelado de intereses en el diseño | Cosmos is a concern-space modeling schema. Cosmos represents concern spaces in terms of concerns, relationships, and predicates. Cosmos divides concerns into two categories, logical and physical. Logical concerns represent concerns viewed conceptually: topics, issues, properties, problems, in general “matters for consideration” [6]. |
| Modelado de intereses en el diseño | A CORE concern has to provide three interfaces to facilitate modular reuse: the variation, customization and usage interfaces. The variation interface clearly describes the diferent functional and implementation variations that the concern encapsulates in a feature model [10]. It specifics the closed variability o�ered by the concern (as in an Software Product Line (SPL) [19]), by describing which features may be selected at the same time. Additionally, the variation interface declares the impact of the diferent variants on high-level stakeholder goals, system qualities, and non-functional requirements in an impact model, expressed with a goal modelling notation [8]. For example, a security concern may o�er various means of authentication, from password-based to biometrics-based solutions, each with difering impacts on the level of security as well as cost and end-user convenience. These qualities have to be weighed, when determining which authentication variant is most appropriate in the current application context. It is the first interface that a concern user interacts with during the reuse process. The customization interface describes how a concern may be adapted to the needs of a specific application and is used when a reusable concern is composed with the application. It is tailored to the user’s selection of features and generated automatically by the CORE tool. Consequently, the customization interface allows a generalized concern to be specialized to the application under development. The customization interface expresses the open variability offered by the concern, similar to what generic classes in programming languages do. A security concern, e.g., may define a generic User as a partial class that needs to be merged with concrete application classes that describe the actual users of the system, e.g., |
| Modelado de intereses | ConMan concern s represent the conventional notion of a concern and can contain other concern model elements. Concern contexts (which specialize concern) can also have associated relationships and constraints. The structure of a concern space can be built up by the grouping of model elements and by explicit relationships and constraints. |
| Modelado de intereses | Concerns can be assigned elements extensionally, intensionally, or by a combination of these. Extensional concerns (e.g., “Printing”) are assigned their elements directly. Intensional concerns (e.g., “Naming”) obtain their elements through evaluation of an associated query. A particular subtype of concern context is composition, which contains a set of elements to be composed and a set of composition relationships. |
| Modelado de intereses | Concerns, relationships, and constraints are all specializations of concern model element , the basic type of ConMan models. As such, al l have fi rst-class status, and al l can be appl ied to one another. Consequently, any concern model element can be loaded into a concern space, grouped i nto a concern, related by a relationship, or const rained by a constraint. These elements also support m ultidimensional separation of concerns (MDSOC) and crosscutting associations. For example, in Figure 1 the feature “Naming” occurs under both the “Features” concern and al so under a “Topic” co ncern to wh ich it is related. Units (discussed below) can also cro sscut m ultiple concerns, relationships, and constraints in a concern space, such as the interface org.eclipse.cme.Entity in Figure 1. |