Modeling Aspects by Category Theory

Serge P. Kovalyov
Technological Institute of Computer Technics
6 Institutskaya st. – 630090 Novosibirsk, Russia
+7-383-333-37-94
kovalyov@nsc.ru

ABSTRACT

A framework for formal analysis of aspect-oriented software development (AOSD) is proposed. AOSD is treated as enriching formal models of programs by traceable refinements that produce their systemic interfaces. Category-theoretic construction of architecture school is employed to formalize this approach. Aspect weaving and separation of concerns are defined as universal constructions. Aspect-oriented scenario modeling is discussed as an example.

Categories and Subject Descriptors

F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—Specification techniques

General Terms

Design, Theory

Keywords

Aspect Oriented Software Development, Architecture School, Traceability, Aspect Weaving

1. INTRODUCTION

Aspect oriented software development (AOSD) [2] aims at explicit separating and composing concerns elaborated in response to particular requirements. Concerns are usually much tangled and scattered across software modules due to intermingling and conflicting nature of requirements. Modularizing them or at least keeping them clearly distinguishable throughout the development process can drastically improve software maintainability. However, AOSD has gained lower level of adoption in software production than modular design up to now. As argued in [15], although the community agrees on what AOSD *is good for*, there are no common paradox-free understanding of what AOSD actually *is*. This is partly caused by lack of a sound uniform metamodel capable to elucidate handling of aspects within modular software development paradigms in a natural way.

We present an attempt to create such metamodel. It is based on the concept of tracing development process steps, since traceability is most compromised by tangling. Concerns are treated as sources of traceable refinements that eventually produce software artifacts. Attaching these refinements to program models allows identifying, composing (weaving), and separating concerns in the course of the development process. In order to provide formal semantics of these operations that doesn't rely on some specific development paradigm we employ category theory. It allows characterizing (mathematical) objects by their interrelations with other objects, avoiding appeal to their "interiors" (by which software artifacts created with different technologies much vary). Constructions in categories produce objects that satisfy extensional ("systemic") criteria: universality (existence and uniqueness of interrelation with similar objects), naturality (independence of multistep interrelation on the way it is traced), and so on. Such objects are usually determined uniquely up to an isomorphism, viz. appropriate abstraction of inessential difference. Visual diagrammatic notation is routinely used to specify constructions.

This motivates employing category theory as paradigm-independent formal tool to reason on software design. The very notion of modularity (the Holy Grail of AOSD) is captured as a class of diagrams that satisfy certain structural properties. Augmenting it with concepts of an interface and of a refinement (expressed in category-theoretic terms) leads to fundamental notion of an architecture school that provides uniform general representation of software design (see [4]). In this setting, we formally present enhancing a design technology by aspects as natural labeling of school constituents by concerns.

We employ scenario modeling as principal example of applying this construction to specific technology. On the one hand, it properly captures operational semantics of aspect-oriented programming that essentially consists in augmenting base program execution scenario with woven aspects. On the other hand, it is a widely used requirements engineering technique empowered with methods for transforming models to architectures, including those based on category theory [14]. Enhancing domain engineering of large-scale distributed systems with aspect-oriented capabilities facilitates rapid incremental non-invasive development [9].

2. ARCHITECTURE SCHOOLS

Category-theoretic approach to formalizing software systems design is being developed since 1970s. According to it, formal models (descriptions) of programs are represented as objects, and actions of integrating (models of) individual components into (models of) systems are represented as morphisms. Composition of morphisms represents multistep integration; identity morphisms represent trivial integrating descriptions into themselves by "doing nothing". Resulting category of descriptions is denoted as *c-DESC*. An example can be found in the area of object-oriented design: classes (formally presented on UML) can be considered as objects in category-theoretic sense, with inheritance relations as morphisms. Another example, particularly pertaining to the topic of this paper, is modeling software systems by execution scenarios. The basic mathematical representation of a scenario is a partially ordered set (poset) [13]. Its elements are atomic events occurred during execution, partially ordered by causal dependence. Actions of scenarios integration are precisely homomorphisms of posets since neither events nor interactions shall be "forgotten" at integration. Hence category **Pos** of all posets and all their homomorphisms plays the role of *c-DESC*.

A system built from multiple components is represented as a *c-DESC*-diagram that consists of the components and their interconnections. Recall that a diagram is a functor of the kind $\Delta : X \to c\text{-}DESC$, where X is a small category (called a schema of Δ). A diagram can be "visualized" as a directed graph of a category X whose points are labeled by *c-DESC*-objects and arrows are labeled by *c-DESC*-morphisms. In order to facilitate category-theoretic reasoning on systems design decisions, we consider diagrams as objects of appropriate category (the (covariant) "super-comma category", see [11]). First, recall that all diagrams with schema X comprise a category, denoted $c\text{-}DESC^X$, whose morphisms are called natural transformations. Recall that a natural transformation from a diagram $\Delta : X \to c\text{-}DESC$ to a diagram $\Sigma : X \to c\text{-}DESC$ is a map $\varepsilon : \mathrm{Ob}\, X \to \mathrm{Mor}\, c\text{-}DESC$ that satisfies the following naturality condition. For every X-objects A, B, and every X-morphism $f : A \to B$ the equality $\Sigma(f) \circ \varepsilon_A = \varepsilon_B \circ \Delta(f)$ holds (in particular, we have $\varepsilon_A : \Delta(A) \to \Sigma(A)$ for every X-object A). A natural transformation ε can be visualized as a "prism" with graphs of Δ and Σ as bases, and arrows ε_A, $A \in \mathrm{Ob}\, X$, as lateral edges. Naturality condition ensures that composition induces minimal amount of auxiliary arrows that cross lateral faces of the prism, i.e. that component-wise integration of a system represented by Δ into a system represented by Σ established by ε respects every interconnection.

A notion of a natural transformation admits straightforward extension to diagrams with different schemas, by adding a functor that adjusts schemas. So a morphism from a diagram $\Delta : X \to c\text{-}DESC$ to a diagram $\Sigma : Y \to c\text{-}DESC$ is a pair $\langle \varepsilon, fd \rangle$ consisting of a functor $fd : X \to Y$ and a natural transformation $\varepsilon : \Delta \to \Sigma \circ fd$. If ε consists of identities (implying that $\Delta = \Sigma \circ fd$) and fd is injective, then Δ is called a subdiagram of Σ (graph of Δ is a labeled subgraph of graph of Σ).

Each object A forms a singleton diagram whose graph consists of the single point labeled by A. Morphisms between singleton *c-DESC*-diagrams are precisely *c-DESC*-morphisms between objects they constitute (in category theory it is said that $c\text{-}DESC^{\mathbf{1}}$, where $\mathbf{1}$ is a singleton category, is isomorphic to *c-DESC*). A morphism from an arbitrary diagram Δ to a singleton diagram is called a cocone. It can be visualized as a diagram in form of a "pyramid" with base Δ and edges directed from points of Δ to the distinct point called the vertex. A colimit of Δ, denoted colim Δ, is a cocone that is universal in a sense that every cocone δ over Δ factors through colim Δ uniquely (i.e. there exists a unique *c-DESC*-morphism c, called a colimit arrow, such that $\delta = \langle c, 1_{\mathbf{1}} \rangle \circ \mathrm{colim}\, \Delta$). Obviously a colimit is determined uniquely up to an isomorphism. Its vertex (called a colimit object) can be thought of as the least "container" that encapsulates all objects of Δ via edges respecting structure of their interconnections. For example, a colimit of a discrete diagram (i.e. the one whose schema has no morphisms except identities) is precisely a coproduct of its objects, which includes all of them (preserving their identity) and nothing more. Even an empty diagram may have a colimit, whose object is precisely an initial object (there exists exactly one morphism from it to any other object). An initial object represents a "componentless" system that can be uniquely integrated into every system (for example, "pure" integration middleware).

These considerations motivate employing a notion of a colimit as category-theoretic abstraction of system synthesis [10]. Existence of a colimit is a necessary condition for a *c-DESC*-diagram to represent a valid system. Clearly it is not sufficient, since various structural rules usually apply (e.g. type constraints in object-oriented design). Diagrams that actually produce systems are called well-formed configurations, and constitute class denoted as *Conf*. In scenario modeling, although any **Pos**-diagram has a colimit, a configuration is considered as well-formed only if its target system scenario is included into it explicitly, without employing any structural computations beyond constructing a coproduct. Such situation is common at requirements engineering where the analyst haven't yet collected enough information to establish powerful structural rules over requirement models. So the class *CPos* of all disjoint unions of **Pos**-cocones is used for *Conf*. Every discrete **Pos**-diagram is well-formed; it represents a disjoint union of parallel (non-interacting) scenarios. To see that "many" other diagrams are ill-formed, consider a pair of **Pos**-morphisms $\{a < b\} \leftarrow \{a, b\} \to \{a > b\}$, where arrows denote bijections $a \to a$, $b \to b$. This diagram exposes a and b as concurrent events, so it is impossible to include both conflicting orderings into the single scenario. In particular, the diagram's colimit object, which is a singleton poset, fails to represent integration result (as well as any other poset).

It is well known that integration capabilities of a component are completely determined by its specially devised "part" called an interface. Interfaces of formal models comprise a category denoted *SIG*; extracting an interface is expressed as a signature functor $sig : c\text{-}DESC \to SIG$. Although it shouldn't be injective (different components can have the same interface), it is required to be faithful, i.e. injective on each hom-set $\mathrm{Mor}(A, B)$ that consists of all *c-DESC*-morphisms with domain A and codomain B (otherwise it would fail to distinguish different ways to integrate a component A into a system B). Realizability of every interface is ensured via existence of a functor $sig^* : SIG \to c\text{-}DESC$ that produces a "default" implementation of every interface I. Specifically, $sig^*(I)$ has I as an interface (i.e. $sig \circ sig^* = 1_{SIG}$) and supports all integration capabilities of I (i.e. functor sig surjectively, hence bijectively, maps a hom-set $\mathrm{Mor}(sig^*(I), A)$ to $\mathrm{Mor}(I, sig(A))$ for every *c-DESC*-object A). In category-theoretic terms functor sig^* is called left adjoint to sig.

There exists a natural correlation between interfaces and configurations. First, conditions $\Delta \in Conf$ and $sig \circ \Delta = sig \circ \Sigma$ shall imply $\Sigma \in Conf$ for any *c-DESC*-diagrams Δ, Σ. This requirement establishes a kind of logical non-contradiction law for interfaces: no interface integration schema can be produced by both systems and illegal conglomerates of components. Second, interface extraction shall be natural with regard to composing systems: every colimit of *SIG*-diagram $sig \circ \Delta$ equals to $sig \circ \mathrm{colim}\, \Delta$ for each $\Delta \in Conf$. In other words, functor *sig* lifts colimits of diagrams from *Conf*. Naturality is actually two-fold: requirements enlisted above imply that functor *sig* preserves colimits of diagrams from *Conf*.

In scenario modeling, an interface of a scenario is a set of occurred events obtained by forgetting their order. Indeed, in order to identify an execution scenario of a component in a scenario of a system, it is necessary and sufficient to identify all events occurred within the component. So the canonical forgetful functor $|\text{–}| : \mathbf{Pos} \to \mathbf{Set}$, where **Set** is a category of all sets and all maps, is used for *sig*. It is easy to see that it satisfies all requirements above. In particular, default realization of an interface is represented by a functor $|\text{–}|^* : \mathbf{Set} \to \mathbf{Pos}$ that turns a set S into a discretely ordered poset $\langle S, =\rangle$ that seamlessly integrates into any scenario.

In addition to system composition, software development process contains steps of modeling individual components known as refinements. The process is commonly viewed as moving along two dimensions: horizontal structuring induced by "component-of" relationships and vertical structuring induced by "refined-by" relationships [10]. All descriptions and all refinements comprise a category, denoted *r-DESC*, with the same class of objects as *c-DESC*. A (trivial) example of a refinement is an isomorphism, so a subcategory of *c-DESC* that contains of all descriptions and all *c-DESC*-isomorphisms is required to be a subcategory of *r-DESC*. The naturality of refinement with regard to composing systems is imposed in the form that a collection of refinements of components constituting a system shall induce a refinement of the system. This can be formally expressed in terms of natural *r-DESC*-transformations, given that every discrete *c-DESC*-diagram is simultaneously an *r-DESC*-diagram. An arbitrary collection of refinements of objects of a *c-DESC*-diagram Δ is precisely a natural *r-DESC*-transformation $\varphi : |\Delta| \to \Sigma$, where $|\Delta|$ is discrete diagram that consists of all objects from Δ, and Σ is a discrete diagram consisting of all refinements results. If $\Delta \in Conf$, then a diagram $\Delta \oplus \varphi \in Conf$ shall exist, that has Σ as a subdiagram (and possibly extra points and arrows), and an *r-DESC*-morphism from a colimit object of Δ to a colimit object of $\Delta \oplus \varphi$.

In scenario modeling, refinement of a scenario consists in replacing atomic events with subscenarios in such a way that the order is fully inherited [5]. Formally, a refinement of a poset X to a poset A is identified with a surjective map $f : A \to X$ that satisfies the condition $\forall x\ \forall y\ f(x) \le f(y) \Leftrightarrow (x \le y \vee f(x) = f(y))$. Notice the swap of source and destination: as we will see below, it is a key to smooth aspect orientation. We will denote the category of all posets and all scenario refinements by *r-Pos*.

A tuple $\langle$*c-DESC*, *Conf*, *sig*, *r-DESC*$\rangle$, that satisfies all conditions enlisted above, is called an *architecture school*, and various examples of schools are considered, in [4]. Of particular interest are schools "over" **Set**, in which *c-DESC* is a concrete category over **Set** in the following sense. Descriptions are sets equipped with some structure (e.g. algebraic structures, topological spaces, etc), integration actions are maps that respect the structure, and *sig* is canonical functor $|\text{–}|$ that forgets the structure. Scenario modeling architecture school $SM = \langle \mathbf{Pos}, CPos, |\text{–}|, r\text{-}Pos\rangle$ is an example of an architecture school over **Set**.

3. ENHANCING DESIGN WITH ASPECTS

Our model of AOSD rests upon representing emergence of aspects in a software design technology as formal transformation of architecture schools. Indeed, AOSD can be generally considered as equipping software artifacts with certain labeling conveniently identifying concerns handled by their constituents. Original motivation of AOSD creators [7] stems from the fact that programming languages are too concise to allow tracing intermingled fragments of source code to their ultimate "goals". Different flavours of AOSD [2] greatly vary in labeling techniques (among which modularization is most welcome) but agree in pursuing transparent traceability, viz. ability to determine exactly what each fragment of a model is included into it for.

A metamodel of traceability proposed in [16] formally demonstrates that tracing is routinely compromised by refinement. A refinement may change the very "nature" of a model, e. g. when implementing a specification by means of a programming language. On the contrary, system composition is able to provide at least partial tracing back to components; difficulties arise at tracing concerns that *crosscut* boundaries of modular architecture (such as security). So ability to trace result of a refinement to its source means that reversing its direction (i.e. category-theoretic dualization) produces a *c-DESC*-morphism, called its *trace*. In order to preserve traceability in subsequent integration of a result into a system, a trace shall have right inverse at the level of interfaces. Indeed, if a refinement $r : X \to A$ satisfies a condition $sig(r^{\mathrm{op}}) \circ s = 1_{sig(X)}$ for some *SIG*-morphism $s : sig(X) \to sig(A)$, then *SIG*-morphism $sig(f) \circ s$ identifies $sig(X)$ in $sig(S)$ for every *c-DESC*-morphism $f : A \to S$.

Obvious example of traceable refinement is a *c-DESC*-isomorphism (recall that a dual to an isomorphism is identified with its inverse which is an isomorphism as well). Non-trivial traceable refinements are obtained as *r-DESC*-morphisms that coherently behave as duals to *c-DESC*-morphisms. Denote by *cr-DESC* the intersection of all such maximal common subcategories of *c-DESC* and *r-DESC*$^{\mathrm{op}}$ that contain all *c-DESC*-isomorphisms.

Definition 1. A *cr-DESC*-morphism t is called a *trace* provided that $sig(t)$ is a retraction (i.e. has right inverse). A *sig*-image of a trace is called a *labeling*. A dual to a trace is called a *traceable refinement*. □

In an architecture school over **Set** every labeling is a surjective map, so a traceable refinement $r : X \to A$ is a total antifunctional binary relation that is conservative with regard to structure. Its action can be described as expansion of points of $|X|$ to sets that comprise partitioning of $|A|$, projecting structural constraints defined on points of $|X|$ to some (possibly none) members of their expansion results. A point of $|X|$ can be considered as a concern that is elaborated by expansion, in accordance with intuitive notion of refinement. For example, in scenario modeling school *SM* *r-Pos*$^{\mathrm{op}}$ is a subcategory of **Pos**; every refinement is traceable, and literally determines a labeling of its target by points of its source. Moreover, refinements allow tracing inclusions of components, viz. integration actions that leave inner structures of components intact. Inclusions are represented by regular **Pos**-monomorphisms; capability to trace them means that for every *r-Pos*-morphism $r : X \rightarrow A$ and inclusion $i : M \rightarrow X$ there exists an inclusion $i' : M \rightarrow A$ such that $r^{\mathrm{op}} \circ i' = i$.

Observe that traceable refinements particularly tolerate configurations. Consider a diagram, called a *push* of Δ by φ, that consists of a diagram $\Delta \in \mathit{Conf}$ and a family φ of arrows directed from distinct points outside of Δ to points of Δ. Obviously a push has a colimit with the same object as Δ. Comprising φ from traces (so that φ^{op} is a natural *r-DESC*-transformation from $|\Delta|$ to a discrete subdiagram of a push), we see that traceable refinements are *non-invasive* with respect to system composition: if a push belongs to *Conf*, then it can be taken for $\Delta \oplus \varphi^{\mathrm{op}}$, and appropriate isomorphism for a refinement of colimit objects. Non-invasive refinements are much appreciated within the context of AOSD. For example, this is obviously the case for scenario modeling school.

These considerations suggest that the AOSD objective can be achieved by equipping descriptions with traceable refinements that produce them, at least at the interface level. Such equipping is precisely the desired aspect labeling. We employ the construct of comma category (see [11]) to formalize it. We will work in specific comma category denoted as $\mathit{sig} \downarrow \mathit{SIG}$. Recall that its objects are pairs $\langle A, a : \mathit{sig}(A) \rightarrow X\rangle$, where $A \in \mathrm{Ob}\ \mathit{c\text{-}DESC}$ and $a \in \mathrm{Mor}\ \mathit{SIG}$. A morphism from an object $\langle A_1, a_1\rangle$ to an object $\langle A_2, a_2\rangle$ is such pair $\langle f : A_1 \rightarrow A_2, b : \mathrm{codom}\ a_1 \rightarrow \mathrm{codom}\ a_2\rangle$ that $b \circ a_1 = a_2 \circ \mathit{sig}(f)$. Denote by *AO* full subcategory of $\mathit{sig} \downarrow \mathit{SIG}$ whose objects are all pairs $\langle A, a\rangle$ in which a is a labeling.

In an architecture school over **Set** aspect labeling $a : |A| \rightarrow X$ of a description A consists in assigning each point of $|A|$ a point of set X that denotes the "name" of the aspect it belongs to. The labeling is essentially (up to an *AO*-isomorphism) an equivalence relation on $|A|$, equivalence classes representing individual aspects. Every such relation turns A into a valid aspect-oriented model, so aspects needn't respect its "modular" structure in any way. *AO*-morphisms are precisely such *c-DESC*-morphisms that preserve this additional equivalence relation. As we will see below, there exists a functor that turns *AO* into a concrete category over **Set**.

Objects of various categories that comprise *AO* can serve as interfaces of *AO*-descriptions, contributing to turning *AO* into full-scale architecture school. Specifically, the software designer have freedom to choose interfaces of aspect-oriented models to be either:

- original non-aspect-oriented models, obtained by functor *mod* that takes an *AO*-object $\langle A, a\rangle$ to a *c-DESC*-object A, for modular design tasks;
- aspect labelings, obtained by functor *asp* that takes $\langle A, a\rangle$ to a, for design and analysis of aspect structure;
- original model interfaces, obtained by functor $\mathit{int} = \mathit{sig} \circ \mathit{mod}$, for specification purposes.

Other options that refine (i.e. can be naturally transmuted to) original interfaces may be available in particular schools.

Refinements and well-formed configurations of aspect-oriented models are constructed by appropriate enrichment of modular "material". Let *tr-AO* be the subcategory of *AO* that consists of all *AO*-objects and all such *AO*-morphisms f that $\mathit{mod}(f)$ is a trace. Further, denote by *str* functor that takes $\langle A, a\rangle$ to codom a. Notice that, given an *AO*-diagram Δ, a diagram $|\mathit{asp} \circ \Delta|$ that consists of labelings of all objects of Δ can be viewed as a natural transformation of $\mathit{int} \circ \Delta$ to $\mathit{str} \circ \Delta$, i.e. $\gamma \circ \langle|\mathit{asp} \circ \Delta|, 1_{\mathrm{dom}\,\Delta}\rangle$ is a cocone over $\mathit{int} \circ \Delta$ for each cocone γ over $\mathit{str} \circ \Delta$. Bearing this in mind, we will call a class *I-Dia* of *SIG*-diagrams *aspect-closed* if for any $\Sigma \in \mathit{I\text{-}Dia}$ an *AO*-diagram Δ satisfies the following conditions provided that $\mathit{int} \circ \Delta = \Sigma$:

- $\mathit{mod} \circ \Delta \in \mathit{Conf}$;
- *SIG*-diagram $\mathit{str} \circ \Delta$ has a colimit;
- every colimit arrow c_Δ, such that $\mathrm{colim}\ (\mathit{str} \circ \Delta) \circ \langle|\mathit{asp} \circ \Delta|, 1_{\mathrm{dom}\,\Delta}\rangle = \langle c_\Delta, 1_{\mathbf{I}}\rangle \circ \mathrm{colim}\ (\mathit{int} \circ \Delta)$, is a labeling;
- for every natural *tr-AO*-transformation $\varphi : \Sigma \rightarrow |\Delta|$ there exists an *AO*-diagram $\Delta \oplus \varphi$, such that Σ is its subdiagram, $\mathit{int} \circ (\Delta \oplus \varphi) \in \mathit{I\text{-}Dia}$, and there exists a *tr-AO*-morphism $t : \langle C^{\oplus}, c_{\Delta \oplus \varphi}\rangle \rightarrow \langle C, c_\Delta\rangle$, where $C^{\oplus}$ is a colimit object of $\mathit{mod} \circ (\Delta \oplus \varphi)$ and C is a colimit object of $\mathit{mod} \circ \Delta$.

Denote by *AO-Int* the union of all aspect-closed classes of *SIG*-diagrams. It allows determining all configurations that retain modularization when constituent components gain labeling by aspects.

Definition 2. Given an architecture school $AR = \langle \mathit{c\text{-}DESC}, \mathit{Conf}, \mathit{sig}, \mathit{r\text{-}DESC}\rangle$, functor *ai* is said to *generate an aspect-oriented architecture school* (AO-school) from *AR*, if a tuple

$$AO_{ai}(AR) = \langle AO, \{\Delta \mid \mathit{int} \circ \Delta \in \mathit{AO\text{-}Int}\}, ai, \mathit{tr\text{-}AO}^{\mathrm{op}}\rangle$$

is an architecture school, and there exists such functor *si* that $\mathit{si} \circ \mathit{ai} = \mathit{int}$. □

Theorem 3. Functors 1_{AO}, *mod*, *asp*, *int* generate AO-schools. □

The proof of the theorem consists in checking that $AO_{ai}(AR)$ satisfies all conditions for an architecture school whenever one of enlisted functors is taken for *ai*. In particular, functor *mod**, which is left adjoint to *mod* and defines inclusion of *c-DESC* into *AO*, takes a *c-DESC*-object A to an *AO*-object $\langle A, 1_{\mathit{sig}(A)}\rangle$. It represents the first step in enhancing a modular design technology by aspects: seed aspect structure coincides with an integration interface. *AO*-descriptions with non-trivial aspect structures emerge upon refining them in the course of AOSD process.

In scenario modeling school *SM*, aspects appear to be precisely labels that, being attached to elements, turn posets into *pomsets* [13]. Labels can be considered as event "names" denoting concerns they handle. Class *AO-Int* coincides with $|{-}| \circ \mathit{CPos}$, which means that all configurations admit aspect orientation. Since every refinement is traceable, all of them are used at constructing $\mathit{tr\text{-}AO}^{\mathrm{op}}$. Functor *mod** endows the discrete labeling on a scenario, equipping each event with a unique label (actually itself).

4. WEAVING AND SEPARATING ASPECTS

Elementary building blocks of aspect-oriented models are known as *aspects*. In an architecture school over **Set**, an aspect is precisely an *AO*-object whose *str*-image is a singleton set, i.e. a terminal **Set**-object (there exists exactly one map from any other set to it). For example, an aspect in scenario modeling is precisely a pomset with all elements labeled with the same label. In order to generalize to arbitrary AO-school, observe that every morphism directed from a terminal object has left inverse (that typically is a trace, so aspects particularly tolerate tracing).

Definition 4. An *AO*-description A is called an *aspect* if $str(f)$ has left inverse for every *AO*-morphism $f : A \to X$. □

Proposition 5. If *c-DESC* has a terminal object **1**, then A is an aspect iff $str(A) = sig(\mathbf{1})$. □

Aspect-oriented program synthesis and decomposition techniques can be formalized as universal constructs in category *AO*. For example, weaving an *AO*-object W (advice) into an *AO*-object B (base) is represented as follows. Weaving rules determine join points in base B at which W is called through appropriate entry points. For example, a program written on an aspect-oriented extension of an object-oriented language, such as AspectJ [3], can be weaved to the base before/after method calls, access operations to fields, exception handlers, etc. In order to specify weaving rules, auxiliary *AO*-object C, called connector, is employed (see [12]) in a way that matching between entry points and join points is described as a pair of *AO*-morphisms $j : B \leftarrow C \to W : e$. Observe that morphism j is usually called a pointcut descriptor [3]. Weaver at first (virtually) produces enough copies of W, one for each join point, with appropriate entry point marked at each copy. Then binding entry points to matching join points establishes the weaving provided that it respects aspect structures of both models. In an architecture school over **Set** the first step of weaving can be formalized as constructing a product $C \times W$; subsequent binding of points is represented as appropriate pushout. These operations admit straightforward generalization to arbitrary school. Recall that a pushout is a colimit of a diagram that has a form of a pair of arrows with the same source. It is used in category theory to generalize set-theoretic operation of identifying “the same” elements in different sets.

Definition 6. An *aspect weaving* of a pair of *AO*-morphisms $j : B \leftarrow C \to W : e$, where B is called *base description*, W is called *description being weaved*, and C is called *connector*, is a pushout of pair $j : B \leftarrow C \to C \times W : \langle 1_C, e\rangle$ provided that it exists (implying that product $C \times W$ exists as well) and is preserved by functor *str*. □

This definition captures intuitive properties of weaving. For example, if B consists solely of join points (i.e. j is an isomorphism), then weaving produces a product $B \times W$. Labeled scenarios (i.e. objects of a category *AO* constructed from constituents of scenario modeling school *SM*) are friendly to weaving. In particular, weaving exists iff the connector “tolerates” concurrency in a sense that it doesn’t impose specific order of executing different aspects of the advice bound to the same join point. Formally, for every $x, y \in mod(C)$ conditions $mod(j)(x) = mod(j)(y)$ and $x \le y$ shall imply that $asp(W)(v) = asp(W)(x)$ for every such $v \in mod(W)$ that $mod(e)(x) \le v \le mod(e)(y)$. This holds for weavers with implicit connectors, such as AspectJ.

The construction of weaving suggests how to extract individual aspects from multiaspect program. The category-theoretic construction of a pullback (dual to a pushout) is employed there. Recall that a pullback is a limit (dual to a colimit) of a diagram that has a form of two arrows with the same destination. A pullback is used to generalize set-theoretic notion of a preimage of a subset: given a diagram $s : S \to A \leftarrow B : f$, where s identifies a subobject S in A, and its pullback $p : S \leftarrow P \to B : q$, morphism q identifies a “preimage” $f^{-1}(S)$. Similarly, a subaspect of an *AO*-object A is essentially a preimage of its aspect structure along a traceable refinement represented by $asp(A)$.

Sound notion of a subaspect allows formal evaluation of modularizing crosscutting concerns, viz. separating them into modular design units. The first step towards modularization consists in *explicating* aspect structure of an *AO*-object as a traceable refinement. Although it may be impossible or ambiguous due to tangling, each nonempty AO-school contains models that allow naturally explicating their aspect structures as well as integration actions.

Definition 7. An *explication* (of aspect structure) of an *AO*-description S is an *r-DESC*-morphism $s : X \to mod(S)$ that is dual to a *sig*-trace and satisfies equality $sig(s^{\mathrm{op}}) = asp(S)$. An explication s is called *universal* provided that for every *AO*-morphism $f : S \to R$ and every explication r of aspect structure of R there exists a *c-DESC*-morphism q, called *explication* of f along r, such that $q \circ s^{\mathrm{op}} = r^{\mathrm{op}} \circ mod(f)$. An (aspectual) *core* of an AO-school is full subcategory of *AO* that consists of all descriptions that have a universal explication. □

Obviously a universal explication is unique up to an isomorphism. Moreover, observe that the explication equality resembles the definition of a natural transformation. This is not a mere coincidence: explicating an *AO*-morphism is actually a functor, and universal explications comprise natural transformation of functor *mod* (reduced on the core) to it. An example of a core *AO*-object is a pair $\langle A, 1_{sig(A)}\rangle$ obtained from a *c-DESC*-object A by functor *mod**.

Once an aspect structure of an *AO*-description is explicated as a refinement, individual aspects need to be extracted from it for subsequent modular development. Partitioning complex models to extractable aspects is known as *separation of concerns*. A key to separation is obtaining *AO*-morphisms with pullbacks as explications.

Definition 8. An *AO*-morphism $m : A \to S$ is called a *subaspect* of a core *AO*-description S if it satisfies the following conditions:

- A is a core aspect;
- explication m' of m is right inverse to a trace;
- explication equality $m' \circ a^{\mathrm{op}} = s^{\mathrm{op}} \circ mod(m)$, where a and s are universal explications of A and S, respectively, determines a *c-DESC*-pullback. □

In an architecture school over **Set**, explication of aspect structure of an *AO*-description S consists in equipping set $str(S)$ with enough “modular” structure to turn map $asp(S)$ into actual trace directed from $mod(S)$. If such equipping is possible, then a candidate subaspect in S can be identified, like in **Set**, by pulling back a (weak) element (i.e. a morphism whose domain rests upon a singleton set) along this trace. An underlying set of the pullback object is precisely an equivalence class of aspect structure equivalence relation. In order for it to form a genuine subaspect, both it and the codomain of the identifying element should be produced from the element’s domain by traceable refinements.

Every core labeled scenario can be partitioned to subaspects. However, the core is rather "small": for example, two linearly ordered aspects executed in interleaving mode cannot be separated from each other. Weaving cannot directly produce interleaving as well. This fact illustrates difficulties encountered at developing even simple client-server distributed systems. Yet every scenario can be labeled by linearly ordered extractable aspects. Their number can be either maximized by applying the functor *mod**, or minimized by identifying so-called sequential subsystems [8]. This fact justifies developing aspect-oriented extensions to traditional programming languages that allow creating only sequential programs.

As an example application of aspect-oriented scenario modeling, consider a distributed measurement system (DMS). As shown in [9], its main execution scenario consists in reiterating the following linearly ordered sequence of data processing (functional) concerns:

$$\textit{measure} \to \textit{store} \to \textit{validate} \to \textit{compute} \to \textit{display}.$$

During system development, each of them is refined to a complex aspect, yet they remain separable. However, infrastructure aspects, such as metadata model, monitoring, and security, are woven to each of them, undermining separation of concerns. So in order to execute different data processing functions on different computers, the infrastructure has to be somehow replicated among them. It is this replication that makes a DMS considerably more challenging to develop than an isolated measurement device.

A glance on results of this section reveals that major contribution of AOSD into software design (in its category-theoretic treatment) consists in employing various kinds of limits (constructions dual to colimits), including a terminal object, products, and pullbacks. Contrast this with traditional modular design that, as presented in Section 2, is based solely on colimits. The root reason of limits to appear is of course the duality between integration actions and traceable refinements, as imposed by Definition 1.

5. CONCLUSION

Our work belongs to the mainstream of applications of category theory to computer science. Their success is due to ability of category-theoretic notions to formally express basic mental patterns of systems analysis, which is the crucial software design activity. In particular, fundamental results were achieved in the area of "categorizing" modular design. However, to the best of our knowledge there are no comparably powerful frameworks suitable to construct and analyze aspect-oriented development technologies. Existing AOSD methods are represented in terms of concrete formal devices difficult to apply beyond specific software development paradigms. Formalisms employed to express aspect-oriented concepts include process algebras [1], model checking [6], architecture description languages [12], graph transformations [17], and so on. In contrast to them, our approach aims at producing aspect-oriented methods suitable for any particular designers' needs by formal transformation of a given modular architecture.

So far presented metamodel is too abstract to be directly applied in software development. Its instances pertaining to major existing design technologies need to be developed and generously illustrated with examples. Such kind of development is a promising area of further research. Much work also has to be done in discovering capabilities and limitations of AO-schools, creating abstract yet powerful aspect weaving and separating techniques.

6. REFERENCES

[1] Andrews J.H. Process-Algebraic Foundations of Aspect-Oriented Programming. Lecture Notes in Computer Science, Vol. 2192, 2001, 187–209.

[2] Aspect-Oriented Software Development. Addison Wesley, 2004.

[3] Colyer A., Clement A., Harley G., Webster M. Eclipse AspectJ. Addison-Wesley, 2004.

[4] Fiadeiro J.L., Lopes A., Wermelinger M. A Mathematical Semantics for Architectural Connectors. Lecture Notes in Computer Science, Vol. 2793, 2003, 190–234.

[5] Glabbeek R.J. van, Goltz U. Refinement of Actions and Equivalence Notions for Concurrent Systems. Acta Informatica, Vol. 37, Issue 4–5, 2000, 229–327.

[6] Katz E., Katz S. Verifying Scenario-Based Aspect Specifications. Lecture Notes in Computer Science, Vol. 3582, 2005, 432–447.

[7] Kiczales G. et al. Aspect-Oriented Programming. Lecture Notes in Computer Science, Vol. 1241, 1997, 220–242.

[8] Kovalyov S.P. Architecture of Time of Distributed Information Systems. J. Computational Technologies, Vol. 7, No. 6, 2002, 38–53. [In Russian]

[9] Kovalyov S.P. Domain Engineering of Distributed Measurement Systems. Optoelectronics, Instrumentation and Data Processing, 44(2), 2008, 125–130.

[10] Lopes A., Fiadeiro J.L. Revisiting the Categorical Approach to Systems. Lecture Notes in Computer Science, Vol. 2422, 2002, 426–440.

[11] Mac Lane S. Categories for Working Mathematician. 2nd Ed. Springer, 2008.

[12] Pinto M., Fuentes L., Troya J.M. DAOP-ADL: An Architecture Description Language for Dynamic Component and Aspect-Based Development. Lecture Notes in Computer Science, Vol. 2830, 2003, 118–137.

[13] Pratt V.R. Modeling Concurrency with Partial Orders. Intl. J. Parallel Programming, 15(1), 1986, 33–71.

[14] Sassone V., Nielsen M., Winskell G. Deterministic Behavioural Models for Concurrency. Lecture Notes in Computer Science, Vol. 711, 1993, 682–692.

[15] Steimann F. The Paradoxical Success of Aspect-Oriented Programming. In Proceedings of OOPSLA'06. Portland, 2006, 481–497.

[16] Vanhooff B., Baelen S. van, Joosen W., Berbers Y. Traceability as Input for Model Transformations. In Proceedings of the 3rd ECMDA-TW. Haifa, Israel, 2007, 37–46.

[17] Whittle J., Jayaraman P. MATA: A Tool for Aspect-Oriented Modeling Based on Graph Transformation. Lecture Notes in Computer Science, Vol. 5002, 2008, 16–27.