Inf Syst Front (2006) 8:115–131
DOI 10.1007/s10796-006-7976-9

Systemic classification of concern-based design methods in the context of enterprise architecture

Pavel Balabko · Alain Wegmann

Abstract Enterprise Architecture (EA) is a relatively new domain that is rapidly developing. "The primary reason for developing EA is to support business by providing the fundamental technology and process structure for an IT strategy" [TOGAF]. EA models have to model enterprises facets that span from marketing to IT. As a result, EA models tend to become large. Large EA models create a problem for model management. Concern-based design methods (CBDMs) aim to solve this problem by considering EA models as a composition of smaller, manageable parts—concerns. There are dozens of different CBDMs that can be used in the context of EA: from very generic methods to specific methods for business modeling or IT implementations. This variety of methods can cause two problems for those who develop and use innovative CBDMs in the field of Enterprise Architecture (EA). The first problem is to choose specific CBDMs that can be used in a given EA methodology: this is a problem for researchers who develop their own EA methodology. The second problem is to find similar methods (with the same problem domain or with similar frameworks) in order to make a comparative analysis with these methods: this is a problem of researchers who develop their own CBDMs related to a specific problem domain in EA (such as business process modeling or aspect oriented programming). We aim to address both of these problems by means of a definition of generic Requirements for CBDMs based on the system inquiry. We use these requirements to classify twenty CBDMs in the context of EA. We conclude with a short discussion about trends that we have observed in the field of concern-based design and modeling.

P. Balabko (✉)· A. Wegmann
Ecole Polytechnique Fédérale de Lausanne,
CH-1015 Lausanne, Switzerland
e-mail: balabko@mail.ru

A. Wegmann
e-mail: alain.wegmann@epfl.ch

Keywords Classification · Concern · Design · Enterprise architecture · Systemic inquiry · Organization level

1. Introduction

This paper presents a classification of *concern-based design methods* (CBDMs) and considers how CBDMs can be used in the context of *Enterprise Architecture* (EA).

EA is a multi-disciplinary approach that enables enterprises to anticipate or react to necessary business or technical changes. In an EA project, the EA team develops *an EA model* (also called enterprise model) that represents the enterprise. The model is usually structured in hierarchical *organization levels*. The highest level typically describes marketing concerns, the middle level describes business processes, and the lower level describes the IT systems. The rational behind structuring EA models with hierarchical levels can be found in Wegmann (2003).

Usually, EA models become very large because they cover a very wide range of concerns from marketing down to IT implementation issues. As a result, EA models are often incomplete, inconsistent or unspecified. Enterprise models exemplify traditional modeling problems such as the one defined by Clarke et al. (1999): "models are often large and monolithic", "designs are too difficult to reuse" and "there is a significant structural misalignment between requirements and code, with design caught in middle". This results in models that are very difficult to understand and therefore these models cannot be used for reasoning about or for designing business and IT systems. The solution to this problem is to make EA models as a composition of smaller, manageable parts: concerns. In order to do this, concern-based design methods

(CBDMs) should be used. In our paper, we compare and recommend CBDMs that can be used in EA methodologies.

The main question of this paper is: What are the CBDMs that can be used in the context of EA and how can they be used? To see how specific CBDMs can be used to specify EA, we have to make a more systematic analysis of concern-based modeling in the context of EA. To make this analysis, we use an approach that was developed for analyzing systems: the *Systems Inquiry* (see http://www.isss.org). *System Inquiry* incorporates four interrelated directions: *systems philosophy, systems theory, systems methodology* and *systems application. Systems philosophy* is concerned with a system's view on the world. This direction studies WHAT exists and HOW we understand what we know. *Systems theory* tries to "recognize system properties, that are general, and structural similarities in different fields". *Systems methodology* studies different methods and "identifies specific, strategies, methods and tools appropriate to work with our system". *Systemic application* studies the application of specific models, methods and tools in some functional context. In our work this context is EA. Therefore, we study the application of specific CBDMs in the context of EA, i.e. how these specific methods can be used to support EA modeling.

We use system inquiry as a base to define the *generic requirements for CBDMs* to be used in EA. These requirements are then used to generate a *list of classified CBDMs* (starting from a *list of existing CBDMs*). This list allows the researchers (i.e. designers of EA methodologies) to select the *list of relevant CBDMs for their* EA methodologies. Using relevant CBDMs improves the structure of the enterprise models and makes them simpler to use. It also improves the traceability between design models at different organization levels.

In our work we consider twenty specific CBDMs and analyze how they can be used in the context of EA. In order to do this we use systems inquiry (Section 2). In Section 3 we use these requirements to see how the specific CBDMs can be used to support EA modeling (the systemic application). In Section 3.1 we start with the brief description of the twenty CBDMs. In Section 3.2 we check how the requirements for CBDMs are satisfied by each CBDM. This results in the *CBDM Requirements Checklist* table. Then we use this table to associate each method with organization levels where the method can be used. This results in the *classification of CBDMs in the context of EA* (Section 3.3). In Section 4 we discuss the generated classification. Section 5 is the conclusion.

2. Requirements for CBDMs in the context of EA based on the "systems inquiry"

In this section, we define requirements for CBDMs in the context of EA based on the four directions of Systems Inquiry. In Section 2.1 we consider *systems philosophy* that defines the main concepts of CBDMs. In Section 2.2 we consider *systems theory* that defines the most general principles of CBDMs. In Section 2.3 we consider *systems methodology* that studies the properties of CBDMs useful in the context of EA. In the summary of each section, we define the specific requirements for CBDMs in the context of EA. These requirements will be used in Section 3 as a basis for the classifications of the CBDMs that we provide in our paper.

The requirements for CBDMs that we give in this section come from different disciplines (philosophy, psychology, system science etc) and have often solid foundations[1] in these disciplines. However, due to the page limit, we mostly omit the description of these foundations and only give practical description of requirements in the context of EA.

2.1. Systems philosophy of CBDMs

In this section, we consider the main modeling elements used in CBDMs and principles that explain how these elements should be used in modeling. CBDMs use many concepts: "view, viewpoint, role, perspective, aspect, subject, etc" (see Nassar et al. (2003)). They share many commonalities. However, as the CBDMs are developed to be used by different specialists, these commonalities disappear behind the difference of names. In this section we consider concepts that serve as a basis for all CBDMs in the context of EA. We also explain how these concepts can be used in the EA design process.

Organization levels. Organization levels reflect the perspectives perceived by different specialists. The idea that the specialists perceive the reality (or the Universe of Discourse—UoD) from multiple perspectives comes from constructivism. Kant claims that "we cannot know things in themselves and that knowledge of the word is possible only by imposing pre-given categories of thought in otherwise inchoate experience" (see social constructivism in Audi (1999)). The same idea can be found in Berkeley's "ubjective idealism." He claims that the perception of the reality (or the UoD) is observer dependant. An observer gets his "abstract ideas" about the UoD by pulling out certain features or qualities from reality and leaving the rest behind. In the context of EA this means that each EA team members has a certain perspective on the UoD, where the perspective explains and justifies in its own way the knowledge about a system. Many design methods (especially in the context of EA) support modeling with organization levels. For example, RM-ODP (ISO/IEC and ITU-T, 1996) defined five viewpoints: Enterprise, Information, Computational, Engineering and Technology; the Zachman framework (Zachman, 1987) defines six architectural

[1] You can find more about the CBDM foundations in our technical report: http://lamspeople.epfl.ch/balabko/foundations.pdf.

perspectives corresponding to the following specialists: a planner, an owner, an architect/designer, a builder, a subcontractor and a user. The concepts of viewpoint and architectural perspective are equivalent to organization level. It is now necessary to explain why these perspectives are perceived as hierarchical. At a given level a system is perceived as a whole and at another level the system is perceived as a set of its parts. The result of this part/whole relationship is the construction of a hierarchical perception of the reality. This idea can be found in Miller (1995) where he shows that the way science has developed is by analyzing the reality as a hierarchy of systems.

EA team members have to understand how to organize the management of model elements at the given organization level. In our work we consider the management of model elements based on compositions of small, manageable parts: concerns. We describe the concepts that EA team members need to represent the concerns at the given organization level. The three following concepts serve as a basis for concern-based modeling: objects, roles and collaborations.

Object is "a model of an entity. An object is characterized by its behavior and, dually, by its state" (ISO/IEC and ITU-T, 1996).

Objects collaborate with each other to reach certain goals.

Collaboration is used to describe a set of collaborating objects. In our work we use collaborations to represent concerns at a certain organization level. A concept similar to collaboration that can be used to represent concerns is *Role Model*, introduced by T. Reenskaug. "*Role Model* specifies the set of collaborating roles along with their state and behavior" (Reenskaug, 1996). Another similar concept is pattern. Patterns can be considered as collaborations "that capture the essential structure and insight of a successful family of proven solutions to a recurring problem" (see http://www.cmcrossroads.com/bradapp/docs/). The structure of patterns can be expressed as the collaboration between several objects. Patterns were introduced by Alexander et al. (1977) and became popular with the "Design Patterns: Elements of Reusable Object-Oriented Software" book (Gamma et al., 1994).

Objects in collaborations play roles.

Role stands for "an abstraction of the behavior of an object" (ISO/IEC and ITU-T, 1996) intended for achieving a certain common goal in collaboration with other objects. A role is usually modeled as a set of actions and state variables related to a given collaboration. Roles are usually used in EA to talk about the business or IT requirements of a system (at the Marketing, Company or IT Application Organizational levels). A concept similar to role is *feature*. This concept is also used for the specification of system requirements. Turner et al. (1998) defines feature as "a clustering of individual requirements that describe a cohesive identifiable unit of functionality". Another concept similar to role is *perspective* defined by Motschnig-Pitrik as "the data structure modeling an object relative to a context" (Motschnig, 1999). Two concepts similar to role at the implementation level are *aspect* and *subject*. Aspect was first defined in AOP by Kiczales et al. (1997) and is used for encapsulating a crosscutting code. It wraps a supplementary code and also stores information about join points and pointcuts.[2] Such join points indicate when the supplementary code should be executed. The term *subject* was proposed in the context of Subject Oriented Programming (SOP) by Harrison and Ossher (from IBM) as an extension of the object-oriented paradigm to address a problem of handling different subjective perspectives on objects to be modeled. According to Harrison and Ossher (1993) the term *subject* means a collection of state and behavior specifications reflecting the perception of the world. SOP uses them to represent a subjective view on objects. Any object can be seen as a composition of several subjects, where each subject can be managed separately.

As we mentioned above, role is a behavior intended for achieving a certain *goal* in collaboration with other roles.

Goals are important in the context of EA because they allow humans to specify what systems do instead of how systems do things. Goals are the products of their foresight and intent (teleological principle). Goals are also used by humans to interpret the functionality of systems (teleonomic principle). In the context of EA, the hierarchy of organization levels is used to model goals: at a given organization level any part of a system (or a system itself) can be considered as a whole that has functionality with a certain result (the goal of this functionality). At the lower level of the organization hierarchy, this functionality is achieved through the collaboration of parts.

Another important concept related to CBDMs is context:

Context. In EA projects, EA team members need to reason about large systems. These large systems can be observed in different situations modeled as contexts. "Contexts provide means to focus on aspects that are relevant in a particular situation while ignoring others" (Motschnig-Pitrik, 2000). Modeling systems in multiple contexts is useful in the analysis of large systems because it allows for dealing with their complexity in a systematic way.
In our work we understand the context as the model of a meaningful situation. The situation represents what is

[2] *Join points* are well-known points in the dynamic execution of a program. And *pointcuts* are sets of join points.

around the system of interest: objects from its environment together with the joint behavior in which they are all involved. I.e. a context can be modeled as collaboration (a set of objects in the environment of the system with their behaviors). Each role of the system of interest relates to given context. Making context explicit makes easier to understand different roles of a system.

Now we describe two principles that we believe are important in the context of EA, from the pragmatic point of view. Both these principles improve the understandability of models:

State-Behavior Holism. In order to design systems, we have to describe the systems' states and behaviors.[3] Many design methods and languages tend to separate systems' states from their behaviors. For example, UML has separate behavior (Sequence Diagrams, Activity Diagrams) and state structure (Class Diagrams as conceptual diagrams). These diagrams are related by means of contracts. The separation of state information from behavior allows for building more compact design diagrams. However, in most interesting systems, state and behavior are highly intertwined and hard to separate. It becomes difficult to understand models of such systems when a modeler has to keep state and behavior, not related explicitly, in mind. The separation of state from behavior information goes against *Holism*, an ability to think about the system as a whole. An example of design methodology that is based on holism is the Object-Process Methodology (OPM) (Dori, 2000). It proposes a method for the complete integration of the systems' states and behaviors within a single graphical model. The holistic representation of state and behavior allows for modeling the life cycle of system attributes (or concepts).

In visual modeling a state-behavior holism principle may play an important role. Instead of the separation of state and behavior information, CBDMs may use the separation of concerns to reduce the complexity of design models. The separation of concerns based on roles, patterns and architectural views is more practical than a separation of systems' state from behavior.

Diagrammatic Representation is important in the context of EA. EA design models should be used by specialists with different backgrounds. These specialists have to communicate in languages that avoid using notations specific only for a certain domain. For example, engineers and business analysts (or other domain experts) should have a common language. This is where the visual modeling helps: "Conceptual graphs and other diagrams have been used successfully as a communication medium between engineers and domain experts" (Sowa, 1999). Diagrams are more convenient than textual notations for modeling EA because they help to make explicit relations, contexts (see Sowa (1999)) and they are based on multiple dimensions (x, y, z, and color dimensions). The Requirements 1 table gives the overview of the requirements identified in this section.

2.2. Systems theory of CBDMs

The systems theory of CBDMs defines the most general properties of CBDMs. In this section we analyze these properties.

Separation of concerns is widely used in software engineering: "Separation of concerns is at the core of software engineering. In its most general form, it refers to the ability to identify, encapsulate, and manipulate only those parts of software that are relevant to a particular concept, goal, or purpose" (Ossher and Tarr, 2001). In our work we consider concern-based methods not only at the IT level (that corresponds to software engineering), but at all organization levels of EA hierarchy. When we mention CBDMs, we refer to all design methods that represent design models as sets of concerns. Each concern is an abstraction that represents a part of a system relevant to a particular concept or a purpose. We believe that there are two common forms of abstraction[4]: generalization and composition. Thus, we consider generalization and composition as the most common properties of all CBDMs.

We start with the definitions of generalization and composition that can be used in the context of EA. Then we continue with the identity relation that plays an important role in the composition of design parts.

Generalization relationship is a form of abstraction that allows a modeler to classify modeling elements in terms of types and instances, where a type is "a predicate characterizing a collection of <X>s" (ISO/IEC and ITU-T, 1996).

Composition relationship is a form of abstraction in the form of a relation between the whole and its parts. There are several influential works that consider composition in the context of design methods (Motschnig-Pitrik and Kaasbøll, 1999a; Kilov, 1999; Steimann et al., 2003; Gerstla and Pribbenow, 1996; Barbier et al., 2003). The diversity of the properties of composition that we can see in the above mentioned works can be explained by different backgrounds of authors aiming to define the primary properties

[3] We use definitions based on RM-ODP (ISO/IEC and ITU-T, 1996): *Behavior: A collection of actions and a set of (sequential) relations between actions. State: A collection of attributes, attribute values and relations between attributes.*

[4] Steimann et al. (2003) considers three main forms of abstraction: composition, classification and generalization. However, in our work we consider classification as the result of generalization: instances become classified based on defined types and subtypes.

Requirements 1: System philosophy

Requirements	Ref#	Name of a concepts or a principle	Description
Support of main concepts	r1.1	Organization level	Views of different specialist that collaborate in an EA project.
	r1.2	Object	The individual, logical component that can play roles in collaborations.
	r1.3	Role	An abstraction of the behavior of an object intended for achieving a certain common goal in collaboration with other roles
	r1.4	Collaboration	A set of collaborating roles along with their state and behavior
Main Principles	r1.5	"Explicit context" principle	CBDMs should support the explicit modeling of context.
	r1.6	"Goal driven" principle	Context should define consequences of the behavior of an object that is placed in this context
	r1.7	State-Behavior Holism Principle	CBDMs should not tend to separate the state and behavior information of a described system
	r1.8	Principle of Diagrammatic Representation	Models built in the context of EA should be based on diagrams that can serve as a means of communication between different specialists.

of composition. Peter Gerstl, for example, mainly considers the composition of physical objects. Frank Barbier and Renate Motschnig-Pitrik look at composition as conceptual modelers of IT systems. Friedrich Steimann considers composition in the context of UML models. Our goal is to find a few properties between all afore mentioned ones (they are too numerous and some of them overlap), which will be useful to define composition in the context of EA. These properties should be applicable at all levels of the EA hierarchy, i.e. we need properties that can be used by specialists coming from different domains. Therefore, we consider the three properties taken from Gestalt (Rescher and Oppenheim, 1955) (a psychological theory that considers how humans perceive composition in general):

Part relationships: "the parts of the whole must stand in some special and characteristic relation of dependence with one another; they must satisfy some special condition in virtue of their status as parts of a whole" (Rescher and Oppenheim, 1955); This requirement reflects the structuralism view on composition: a whole is derived from its parts.
Composition structure: "the whole must possess some kind of structure in virtue of which certain specifically structural characteristics pertain to it". The "… structural features of wholes are of interest because one important idea covered by the term 'whole' is that of a structured organization of elements. A structured whole in this sense involves three things: (1) its *parts*, (2) a *domain of 'positions'* that these parts 'occupy' (this need not necessarily be spatial or temporal, but may have any kind of topological structure whatever), and (3) an *assignment* specifying which part occupies each of the positions of the domain" (Rescher and Oppenheim, 1955). This requirement also reflects the structuralism view on composition: it defines how the whole is composed from its atomic parts.
Emergence: "the whole must possess some attribute in virtue of its status as a whole, an attribute peculiar to it and characteristic of it as a whole" (Rescher and Oppenheim, 1955); This reflects the Gestalt view on composition that attributes of wholes cannot be derived from attributes of parts. The similar idea can be found in Kilov (1999): "There exists also at least one property of a composite instance independent of the properties of its component instances."

The "part relationship" requirement for composition states that parts of a whole are mutually dependant. This dependence is specified by relations between parts. In the context of EA, there is a dependence relation that is especially important: identity.

Identify: In its classical form, *identity* is a relation between two elements (in the form: "$a = b \leftrightarrow \forall F(Fa \rightarrow Fb)$"). This means that the identity of a and b is implied by their sharing of all their properties. If a and b are identical relative to one property (predicate), but not to another then this relation is called *relative identity*. If we read the definition of identity from right to left, we can see that if the two elements are indiscernible, then this means that these two elements

Requirements 2: System theory

Requirements	Ref #	Name of a concept or a principle	Description
Support of Generalization	r2.1	Generalization	A form of abstraction that allows a modeler to classify modeling elements in terms of types and subtypes
Compatibility with Composition principles	r2.2	Part relationships	"The parts of the whole must stand in some special and characteristic relation of dependence with one another; they must satisfy some special condition in virtue of their status as parts of a whole."
	r2.3	Composition Structure	"The whole must possess some kind of structure in virtue of which certain specifically structural characteristics pertain to it."
	r2.4	Emergence property	"There exists also at least one property of a composite instance independent of the properties of its component instances."
Compatibility with identity principles	r2.5	Explicit identity modeling	Identity relationship between objects should be specified explicitly in the composition of concerns.
	r2.6	Explicit cross-level traceability	CBDMs that support modeling at several organization levels, should support the cross-level identities or traceability.
	r2.7	Support of relative identity	Relative identity should be used.

are identical. The indiscernibility of two elements can be explained by the fact that these two elements model the same entity in the UoD. Based on this observation there is another definition of identity that states: two elements are identical if they reference the same entity in the UoD. In linguistics there is a similar concept called *coreference*[5] that expresses the relation between signs.

In the context of EA we see the following requirements for modeling identities:

Explicit identity modeling: In the system design using CBDMs, the identity relationship between objects should be specified explicitly in the composition of concerns.

Explicit cross-level traceability: CBDMs that support modeling at several organization levels, should support the cross-level identities or traceability.

Support of relative identity: In the context of EA "relative identity" should be used. This means that identities should be specified in the form: "x is the same F as y", where F is a predicate defined at a certain organization level.

The Requirements 2 table gives the overview of the requirements identified in this section.

2.3. Systems methodology of CBDMs

Systems methodology identifies and studies specific methods and tools appropriate in a given context. In our paper we study specific CBDMs in the context of EA. In EA an enterprise is represented as the composition of multiple organization levels reflecting views of different specialists. Any organization level may use CBDMs specific for this level. Therefore almost any CBDM can be used in the context of EA (at a certain organization level). We provide a list of all CBDMs considered in our paper in the next section, where we consider how these methods can be used in the context of EA.

Instead of identifying specific CBDMs that can be used in the context of EA, in this section we draw attention to some important properties of the EA methodologies. An EA methodology includes the three main development activities that should be used for each organization level (see Fig. 1 where we show an example with two organization levels):

- Multi-level modeling: identifying existing entities (such as processes, goals and tools) in the UoD and representing them in the multi-level enterprise model (as-is model). Each organization level represents entities that are interesting to a given specialist.
- Multi-level evolution: modifying the as-is model to fill the gap between what exists and what should exist to achieve a certain goal of a project. The result of modification is a to-be model at each organization level.

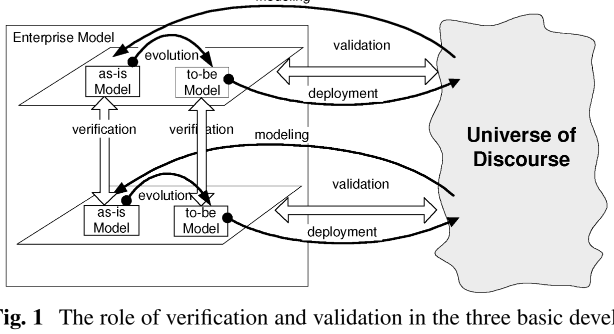


Fig. 1 The role of verification and validation in the three basic development activities (modeling, evolution and deployment)

[5] Coreference is the reference in one expression to the same referent in another expression.

- Multi-level deployment: transforming the TO-BE model into new or modified entities (processes and tools) in the UoD.

The three development activities can be repeated several times. When the first iteration is finished (i.e. the deployment has been completed), then the to-be model takes the place of a new as-is model, the evolution and deployment are done in the second iteration and so forth.

The result of the three development activities is available only after the model deployment (when a model becomes executable). In many cases, checking as-is and to-be models before the deployment can save the time and resources of an enterprise. There are two processes that are used to check the models: *validation* and *verification* (see Fig. 1). The Requirements 3 table gives the overview of the requirements identified in this section.

3. Classification of CBDMs

In this section, we consider twenty CBDMs and analyze how these methods can be used in the context of EA. We begin this section with the list of twenty CBDMs that we classify in our paper (Section 3.1). We check how the requirements for CBDMs (from Section 2) are satisfied by each CBDM from Section 3.1. This results in the *CBDM Requirements Check* table (Section 3.2). We use this table to associate each method with organization levels where this method can be used. This results in the *classification of CBDMs in the context of EA*.

3.1. CBDM requirements check

We begin with a brief description of CBDMs that we consider in our paper. These CBDMs are sorted based on their application domains. The CBDM application domains represent our understanding of the positioning and goals of CBDMs as presented by their authors. We use the following application domains: generic methods, DB and conceptual modeling methods, business process related methods, design pattern modeling methods, methods based on architectural connectors, programming and modeling for programming methods.

The list of CBDMs in this section is not exhaustive, but we included several influential methods in each domain. For each method we describe its goal, main concepts and principles. We also show how these concepts and principles relate to the ones from the previous section.

3.1.1. Generic methods

Generic CBDMs use general terms to describe the deliverables of EA. These methods usually cover several EA organization levels. Generic CBDMs can be used together with specific CBDMs: in this case generic deliverables at a given organization level can be substituted with specific deliverables, more adequate for a certain specialist. We have considered the following generic CBDMs:

- **Object-Oriented role analysis and modeling OORam:** a generic method developed by **T. Reenskaug**

  Goal: Modeling complex systems as structures of interacting objects.

  Concepts and Principles: *object* (r1.2), *role* (r1.3), *role model* (or collaboration, r1.4), *synthesis of role models* (or composition) based on generalization of behavior (r2.1), behavior substitution (or behavior identity, r2.5) and sequential composition (part relationship r2.2).

  Description: OORam is a highly successful generic methodology applied mostly in software engineering and with some opportunities for business analysis (Reenskaug et al., 1996). This is the first work that systematically studied and used the concept of role, role model and composition of role models. OORam

Requirements 3: System methodology

Requirements	Ref#	Name of a concept or a principle	Description
Support of the methodological properties	r3.1	Verification	Compares two levels of system specification for proper correspondence.
	r3.2	Validation	Determines how well the model describes the actual behavior of the system. *Consistency checking* and *the model simulation* are the most common formal techniques that can be used for model validation. Consistency checking guarantee that a model does not have contradictions. Simulation allows a modeler to consider specific scenarios and interpret them. System design verification can also be achieved through cross-level *correctness preserving transformations*.
	r3.3	Executable	The possibility to execute deployed models.

pays attention to the *safe synthesis* that guaranties the integrity of the base model activities in the derived models.

Tool: "OOram Professional" was developed. However, this case tool is not supported any more.

Advantages: Simple, visual (r. 1.8), encourages model and software reuse.

- **Systemic Enterprise Architecture Methodology (SEAM):** a generic method developed by **A. Wegmann**

Goal: Systemic hierarchical system modeling in the context of EA.

Concepts and Principles: *organization level* (r1.1), *object* (r1.2), *role* or localized action (r1.3), *joint action* (or collaboration, r1.4), the explicit modeling of context and environment (r1.5), visual representation of post-conditions (state-behavior holism, r1.7).

Description: SEAM (Wegmann, 2003) is a new generic method in the field of EA. This method has solid theoretical foundations and aims to be used by practitioners in the field of EA. It supports many properties that we have identified in Section 2.

Tool: SEAMCad (see http://lamspeople.epfl.ch/le/SEAM tool/SeamCadl/toc.htm) is under development.

Advantages: Visual (r.1.8), systemic, good support for the hierarchical modeling, proposes a unified otology for all EA levels and, therefore, can be used as basis for the development of the specific EA methods.

- **View Points Framework:** a generic framework proposed by **B. Nuseibeh** et al.

Goal: To make explicit inter-viewpoint relations.

Concepts and Principles: *viewpoint* (organization level, r1.1), viewpoint *relationship rules* (part relationships, r2.2), *agent* (object, r1.2), correspondence between types in different viewpoints (explicit traceability, r2.6).

Description: ViewPoints Framework is a viewpoint method construction framework that defines how to construct methods that integrate multiple viewpoints. It suggests how to create viewpoint templates and use these templates for building hierarchical specifications. This framework addressed "the notion of interViewpoint communication as a vehicle for Viewpoint integration" (Nuseibeh et al., 1993). Inter-viewpoint communications are based on the inter-viewpoint rules defined in viewpoint templates. The paper (Nuseibeh et al., 1993) that describes the basics of ViewPoints framework in 2003 got the Most Influential Paper Award in ICSE, one of the leading conferences in Software Engineering.

Tool: The direct successor of the ViewPoints framework is xlinkit framework (Nentwich et al., 2003) (see http://www.xlinkit.com) that aims in consistency checking and repairing of software engineering artifacts. The xlinkit framework has a case tool of the same name.

Advantages: Helps to make relations between viewpoints more formal and, therefore, allows for the consistency checking between viewpoints.

- **Generic Context Framework:** a generic framework proposed by **R. Motschnig-Pitrik**

Goal: To decompose Information Bases (IBs) using contexts.

Concepts and Principles: *information units* (all possible model elements such as objects, attributes, methods); context (something like a role in our terminology, r1.3) refers a subset of information units, i.e. context is used to split IBs; *authorization* (specify user rights for the execution of actions), *change propagation* (specify part-relationships between contexts, r.2.2) and *owner* (a user who create a context).

Description: This framework originally proposed in Mylopoulos and Motschnig-Pitrik (1995) is inspired by data base views. It considers general abstractions for partitioning information bases with contexts. Later this framework was presented as a generic one for any modeling notation (Motschnig-Pitrik, 2000). It proposes a context to be a "first-class citizen" associated with properties and behavior of objects. This framework was applied in the context of OO modeling, which resulted in the extension to UML that supports the modeling of views (Motschnig-Pitrik, 2000b).

Advantages: This generic framework is strongly influenced by the data base views and, therefore, it is most appropriate for the development of IBs specific frameworks.

3.1.2. DB and conceptual modeling

A group of methods that make an accent on the information modeling for information systems.

- **Lodwick:** a modeling language proposed by **Friedrich Steimann**

Goal: To define the semantics of roles in conceptual modeling.

Concepts and Principles: *object* (r1.2); *role* (r1.3); *role and object type hierarchies* (generalization, r2.1); *static model* or invariant model; *dynamic model* that specifies all possible sequences of model snapshots.

Description: Lodwick is one out of few languages based on logic (order-sorted logic more precisely). It "is intended to be an exploratory language for object-oriented modeling with roles at the conceptual level" (Steimann, 2000).

Advantages: Lodwick provides a good formal definition of roles and their properties that can be used to explain the semantic of roles in CBDMs. Lodwick allows for the expression of many propositions about roles such as: "roles can play roles," "a role can be transferred from one object to another," "an object my play the same role several times" etc.

- **Object-Role Modeling (ORM):** a modeling method proposed by **Halpin**

  Goal: To simplify the conceptual design by using natural language, intuitive diagrams and representing information in terms of simple or elementary facts.
  Concepts and Principles: *object* (r1.2); *role* (r1.4); *facts or n-ary predicates* (resemble to collaborations, r1.4) that can be regarded as sentences with one or more "object-holes" where each hole represents a *role; constraints* between roles (part relationships, r2.2); role *generalization* (r2.1); visual and simple (r.1.8)
  Description: ORM is a method for performing information analysis and design at the conceptual level. It is considered as an alternative to Entity-Relationship — a group of conceptual modeling methods. "Early versions of ORM were developed in Europe in the mid-1970s (for example, binary relationship modeling and Natural Language Information Analysis Method (NIAM))" (Halpin, 2001). ORM is one of the methods appropriate for EA because it "simplifies the design process by using natural language, as well as intuitive diagrams which can be populated with examples" (Halpin, 2001).
  Tool: Microsoft Visio for Enterprise Architects; in addition, ActiveQuery can be used for querying ORM models.
  Advantages: ORM diagrams are simple and can be used by many different specialists (and especially business people). ORM models can be converted to database schemas, ER and UML diagrams.

- **Metapattern:** a conceptual modeling method proposed by **P. Wisse**

  Goal: Information analysis and design that uses context and time as first-class modeling elements.
  Concepts and Principles: *object* (r1.2); *context* (r1.5); *intext* (or role, r1.3); *information objects* (or role attributes); *pointer information object* (or identity relation, r2.5);
  Description: Metapattern (Wisse, 2000) uses a simple visual notation in the form of a directed graph where nodes represent objects and edges represent contextual relations between objects. Any object in Metapattern can be defined only in the context of another object using a contextual relation. The directions of contextual relations show the order of nested contexts.
  Tool: KnitbITs is a commercial tool for prototyping. It assists strategic planning for enterprise engineering.
  Advantages: Highly focused analysis tool that provides a formal treatment of context; a visual notation (r1.8) for the representation of roles, contexts and relations between them for modeling at the high level of abstraction.

- **VBOOL and VUML:** a methodology and modeling language proposed by **Nassar** and others

  Goal: To introduce the notion of a user view associated to every actor of a system.
  Concepts and Principles: *actor* that interact with a system (object, r1.2); *view* of an actor on a system (role, r1.3); *view dependencies* (part relationships, r2.2); view extension (specialization/generalization, r2.1).
  Description: View Based Object-Oriented Methodology Language (VBOOL) (Marcaillou et al., 1994) is an OO language based on multiple inheritance (inspired by Eiffel) with the explicit notion of user views. View based Unified Modeling Language (VUML) (Nassar et al., 2003) is an extension of UML that introduces user views associated to every stakeholder of a system. VUML was inspired by VBOOL.
  Tools: VBOOL interpreter.
  Advantages: Allows for the specification of user needs and access rights in a visual way (based on the extension of UML class diagrams).

3.1.3. Business process (BP) related methods

BP related methods aim at the analysis and design of workflows and processes in an organization.

- **Role Activity Diagrams (RAD);** a visual language proposed in by **Holt** and enriched by **Ould**

  Goal: To express coordinated human behavior.
  Concepts and Principles: *role* (r1.3); *actor* (or objects, r1.2); *interaction* (or detailed collaboration, r1.4); *goal* of a role in collaboration (r1.6).
  Description: RADs are based on concepts proposed by Holt et al. (1983). Later RADs were improved by Ould (1995). RADs were a major feature of the Business Process Reengineering movement in the 1990's. RADs are similar to UML Activity Diagrams (ADs) with swim lanes. They are different in the visual notation and the model elements that can only be modeled in RADs (goals, data flows, interactions between roles). These model elements make RADs more comfortable for business process modeling (see Odeh et al. (2002) for details

on the comparison). RADs are based on the underlying Petri-Nets formalism.

Tools: RADRunner (see http://www.rolemodellers.com) is a commercial product with the underlying XML dialect, Playwright, that allows for the integration with web technologies.

Advantages: Allows business processes to be expressed visually (r1.8) at a high-level of abstraction. RADs are well known and supported with a powerful tool.

- **Role Interaction Nets (RINs);** a visual language proposed by **Singh** and **Rein**

Goal: To express coordinated human behavior.

Concepts and Principles: *role* (r1.3); *actor* (or objects, r1.2); *interaction* (or detailed collaboration, r1.4); *output* (or goal of a role in collaboration, r1.6).

Description: RINs (Singh and Rein, 1992) are very similar to RADs: they both use similar concepts, principles and based on the Petri Nets formalism. However, RINs did not advance as much as RADs. As a result, RINs are not referenced in recent research publications.

Tools: Deva (Rein, 1993) is a role-based collaborative tool that allows people to coordinate their work. This tool does not exist anymore.

Advantages: Allows business processes to be expressed visually (r1.8) at a high-level of abstraction.

3.1.4. Design pattern modeling methods

Design patterns are proven solutions to recurring problems. Patterns in the field of object-oriented analysis and design were first studied by Gamma in his book "Design Patterns: Elements of Reusable Object-Oriented Software" (Gamma et al., 1994). These patterns are usually described using class diagrams. Some researchers go beyond class diagrams and develop new languages for the specification of design patterns. Here we overview one approach, relevant to the subject of our paper, that specifies patterns as a set of collaborating objects.

- **Role Diagrams:** proposed by **Dirk Riehle**

Goal: To specify design patterns as a set of collaborating objects and show how design patterns are applied to objects.

Concepts and Principles: *class*, i.e. a set of objects of a given type (r1.2); *role* (r1.3); *collaboration* (r1.4); role and class *generalization* (r2.1); *composition constraints* (r2.2); *a role diagram* represents a set of collaborating roles together with composition constraints, generalization and composition relations between roles; *a class model* shows how classes play roles from role diagrams.

Description: Role diagrams are the major contribution of the Riehle's research work. They are quite simple and powerful for solving concrete design problems. Role diagrams where inspired by OOram and the work of Alexander et al. (1977).

Advantages: Convenient for the description of design patterns in a visual way (r1.8) (Riehle, 1996), (Riehle, 1997); convenient for the design of new OO Frameworks (Riehle and Gross, 1998).

3.1.5. Architectural connectors

Methods based on the architectural connectors (we take this name from Fiadeiro and Lopes (1997)) aim to separate object essential behaviors (services provided by this object) and object interactions. This separation allows for the explicit representation of object interactions in the form of contracts, communication objects or connectors.

- **CDE from ATX Software:** proposed by **J. Fiadeiro, L. Andrade** at al.

Goal: To separate basic business components from coordination elements (business rules) managed by configuration elements (business policies).

Concepts and Principles: *components* or basic business blocks (i.e. objects, r1.2); *coordination contract* (something like collaboration, r1.4); *configuration elements* or business policies (goal of a coordination contract, r1.6); computation, coordination and configuration layers (organization levels, r1.1).

Description: It uses coordination contracts to represent explicitly the rules that determine Java object interactions (Andrade and Fiadeiro, 1999; Fiadeiro and Lopes, 1997). Coordination contracts support interactions to be externalized as first-class citizens, allowing for the online deployment of coordination contracts.

Tool: CDE is the Java-based Coordination Development Environment (CDE). CDE also allows for the simulation of the coordination mechanism using an animation tool integrated in CDE.

Advantages: Allows for the dynamic reconfiguration of a system caused by the changes of business policies and rules.

- **Sina:** a programming language proposed by **Mehmet Aksit** at al.

Goal: To structure, abstract and reuse object interactions.

Concepts and Principles: *object* (r1.2); *Abstract Communication Types, ACTs* (represent object collaborations, r1.4, or inter-object constraints, i.e. part relationship, r2.2); *composition filters* are used to intercept and redirect messages from objects to ACT objects.

Tool: Sina (Aksit and Tripathi, 1988) is a programming language with the explicit representation of object interactions in the form of Abstract Communication Types (ACTs) (Aksit et al., 1993). ACTs represent explicitly complex communications between objects, such as distributed algorithms, coordinated behavior, inter-object constraints.

Advantages: Makes the complexity of programs manageable by moving the interaction code to separate modules; can implement the synchronization among participating objects.

- **ConcernBASE:** a language and method proposed by **M. M. Kandé**

Goal: To provide a software engineering approach that allows for the separation of concerns in software architecture descriptions.

Concepts and Principles: *components* (i.e. objects, r1.2); *connectors* (similar to collaborations, r1.4); connector consists of two or more *connection points* (similar to a role in our terminology, r1.3) and one connection role or a communication protocol between connection points.

Description: ConcernBASE (see http://lgl.epfl.ch/ research/concernbase/index.html) is a concern-based and architecture-centered software engineering method. To represent object interactions, ConcernBASE uses connectors. "A connector is an abstraction that explicitly represents a locus of definition for component interconnections and communication responsibilities" (Kandé and strohmeier, 2000).

Tool: The Concern BASE Modeler tool is an integrated tool for the development of UML-based architectural descriptions using the Concern-BASE approach. This tool is currently under development.

Advantages: UML-based; complements current Architecture Description Languages (ADLs) with the separation of concerns mechanism.

3.1.6. Programming and modeling for programming

The common goal of methods from this section is to implement roles as source-code entities and then assign these roles to objects.

- **Methods for Implementing Roles** proposed by **D. Notkin and M. VanHilst**

Goal: To implement roles as C++ code entities and compose them into classes using separate composition statements.

Concepts and Principles: *roles* in the form of C++ templates (r1.3); *classes* (that instantiate objects, r1.2); *composition statements* that specify how roles are composed with classes (something like composition structure, r2.2); *roles/responsibility matrix* where rows represent collaborations (r1.4) and columns represent classes.

Description: This method implements roles as source code entities using C++ class templates defined in a stylized way (VanHilst and Notkin, 1996). These templates are composed into C++ classes at compile time using separate composition statements. The composition statements are based on the roles/responsibility matrix that is used to define relations between roles' attributes and methods, and the order in which these roles are composed.

Tool: C++: it uses the features associated with class templates in C++.

Advantages: Improves C++ code maintainability and reuse; requires no special tools.

- **Subject-Oriented Programming (SOP):** proposed by **Harrison and Ossher**

Goal: An extension of OOP that addresses a problem of handling different subjective perspectives on objects to be modeled;

Concepts and Principles: *object* (r1.2); *subject* is a collection of object's state and behavior specifications related to a particular concern (like role, 1.3); *concern* is an expectation or a goal that a stakeholder has on a system (can be represented as collaboration with a goal, r1.4); *composition specification* (specifies part relationships, r2.2) that includes *composition relationship* (or identity relationship, r2.5) and *integration specifications* (tells how the identical elements should be treated: merged, overwritten or selected depending on a context).

Description: Subject Oriented Programming is a programming paradigm proposed by Harrison and Ossher (from IBM) (Harrison and Ossher, 1993). SOP uses subjects to represent a subjective view on objects. Any object can be seen as the composition of several subjects, where each subject can be managed separately. The most known implementation of SOP is the Hyper/J language (see http://www.alphaworks.ibm.com/tech/hyperj).

Tools: Hyper/J—a Java application that allows for the composition of conventional Java classes according to composition rules. Hyper/J composes Java class files based on the special options file. This options file indicates files that participate in a composition, how parameters or actions with the same names should be treated, and other complimentary information.

Advantages: SOP is as generic programming paradigm that allows for the separation of concerns. The separation of concerns helps to trace requirements (in the form of use-cases or features), to code (in the form of programming concerns); SOP improves comprehensibility; SOP has an open-ended semantics of composition that allows for the definition of complex composition patterns.

- **Aspect-Oriented Programming (AOP):** programming language introduced by **Xerox**

  Goal: To localize code that is scattered across several classes.
  Concepts and Principles: *source code* (instantiates objects, r1.2); *advice* is the cross-cutting code to be added to the source code of objects (something like role, r1.3); *pointcuts* specify the composition structure of a source code with advices, r2.3; *aspect* is the combination of pointcuts and advices.
  Description: Aspect-Oriented Programming was named by Gregor Kiczales and his group (Kiczales et al., 1997). It was based on the ideas of adaptive programming that were developed in the early 90th (Lieberherr, 1992). AOP paradigm introduces a new concept to OOP called *Aspect* for encapsulating a crosscutting code. There are many examples of aspects: error checking and handling, synchronization, context-sensitive behavior, performance optimizations, monitoring and logging, debugging support, multi-object protocols.
  Tools: The first version of the AOP language and language processor, AspectJ, that interleaves or weaves objects and aspects was done by Gregor Kiczales, Crista Lopes and other researchers at Xerox PARC. Now AspectJ has evolved into a powerful AOP framework. The information about other AOP frameworks can be seen on http://aosd.net/technology/practitioners.php.
  Advantages: Modularization: redundant code can be placed in aspects; Concentration on the business logic: security, synchronization and other non-business concerns can be handled with aspects; Comprehensibility; Debugging: debugging code can be outside of the main code; Acceptance in industry: integration with developer frameworks such as JBoss (JBossAOP), NetBeans and Eclipse.

- **Aspect-Oriented Design;** design method proposed by **Elizabeth A. Kendall**

  Goal: A role-based design method and its implementation in AOP.
  Concepts and Principles: *classes* (that instantiate objects, r1.2); *aspects* and *roles* (r1.3); *role models* (i.e. collaborations, r1.4).
  Description: Kendall (1999) proposes an aspect-oriented design method that can be implemented using AOP. This design method is specified with role diagrams proposed by Riehle (1997). It also uses the graphical notation of role composition inspired by Kristensen (1995). E. Kendall considers different options for mapping roles from role diagrams to aspects in AOP and discusses the advantages and problems of these options. In Kendall (1998) E. Kendall discusses how goals of a system can be specified and assigned to roles that appear in a model.
  Advantages: Simple visual (r1.8) notation that allows for choosing different options of role model implementations; intuitive for the implementation of design patterns with AOP.

- **Stratified Architectures:** a method and architecture proposed by **Colin Atkinson** and **Thomas Kühne**

  Goal: To provide a method for hierarchical modeling that uses concern-based abstractions.
  Concepts and Principles: *strata* or level of abstraction (r1.1); *object* (r1.2); *object interaction* (i.e. collaboration, r1.4); *interaction refinement* based on the introduction of new system concerns.
  Description: Colin Atkinson and Thomas Kühne in Atkinson et al. (1999) propose a systematic organization of concern-based models in a form of hierarchical structure. This structure allows for the abstraction of "system details step by step so that certain aspects [concerns] can be ignored at a sufficiently high level of abstraction" (Atkinson et al., 1999).
  Advantages: Comprehensibility: stratified architectures make explicit why and where (at which level) a particular concern is introduced in a system, and what are the implication of this concern on the system's overall structure; System redesign becomes easier.

- **OO Modeling with roles:** a methods proposed by **Kristensen**

  Goal: Modeling of perspectives based on the aggregation of roles.
  Concepts and Principles: *perspective* (similar with organization levels, r1.1); *intrinsic object* (or object, r1.2); *role object* (or role, 1.3); *generalization* (r2.1); *emergent methods and attributes* (r.2.4); method and attribute dependencies (or part relationships, r2.2) such as hereditary, aggregated, modified methods.
  Description: Bækdal and Kristensen use roles to specify systems from different perspectives (Bækdal and Kristensen, 1999); (Kristensen, 1995). Each perspective models a system with its own aggregation hierarchy: a perspective defines roles at the lowest level of abstraction and then aggregates them in a form of a hierarchy.

Table 1 CBDM requirements checklist table

Year	Method/ Language Name	Authors	Tool or Language	System Philosophy								System Theory							System Methodology			Systemic Application
				Main Concepts				Main principles				generalisation	Composition			Idenity principles			Verification	Validation	Executable	
				object	role	collaboration	Organization levels (views)	"Explicit Context"	"Goal Driven" contexts/roles	State- Behav. Holism	diagrammatic representation		Emergent Properties	Part relationships	Composition Structure	Explicit Identity relat.	Explicit Traceability	Relative Identity				
approx. 1975	Object Role Modeling	Halpin	FORM in MS Visio for Enterprise Architect	Y	Y	Y					Y	Y		Y							supports conceptual queries	IT, Business
1983	RAD	Holt at al	RADRunner	Y	Y	Y		Y	Y		Y			Y						Simulation		Business
1988	Sina	Aksit	Sina language	Y	Y	Y						only through delegation		Y	Y						Y	IT
1992	RINs	Singh Rein	Deva		Y	Y		Y	Y		Y	Y		Y						Simulation		Business
1993	SOP	William Harrison	Hyper/J	Y	Y	Y						Y		Y		Y					Y	IT
1993	ViewPoints Method Construction Framework	B. Nuseibeh	xlinkit				Y					Y		Y			Y		Consistency Checking	Consistency Checking		IT, Business
1994	VBOOL, VUML	Marcaillou at al.	VBOOL, VUML	Y	Y						Y	Y		Y								IT, Business
1995	Generic Context Framework	R. Motschnig-Pitrik	extention to UML	Y	not in the same meaning			Y			Y	Y		Y		Y						IT, Business
1995	OO Modeling with Roles	B. Kristensen		Y	Y	Y	Y				Y	Y	Y	Y		Y						IT
1995	OORam	Reenskaug	TASKON/ OOram	Y	Y	Y	Y	Y	Y		Y	Y		Y			Y					IT, Business
1996	AOP	Kiczales	AspectJ	Y	Y							Y			Y						Y	IT
1996	Methods for Implementing Roles	Notkin	C++	Y	Y	Y						Y			Y						Y	IT

(*Continued on next page*)

Table 1 (*Continued*)

Year	Method/ Language Name	Authors	Tool or Language	System Philosophy								System Theory							System Methodology			Systemic Application
				Main Concepts				Contexts		Some Principles			Composition			Idenity principles						
				object	role	collaboration	Organization levels (views)	"Explicit Context"	"Goal Driven" contexts/roles	State- Behav. Holism	diagrammatic representation	generalisation	Emergent Properties	Part relationships	Composition Structure	Explicit Identity relat.	Explicit Traceability	Relative Identity	Verification	Validation	Executable	
1996	Role Diagrams	Dirk Riehle		Y	Y	Y		Y			Y	Y		Y								IT, Business
1999	CDE from ATX Software	J. Fiadeiro, L.Andrade at	CDE	Y	Y	Y	Y	Y	Y			Y								Animation of the run-time behavior	Y	IT
1999	Aspect-Oriented Design	Elizabeth A. Kendall		Y	Y	Y		Y	Y		Y	Y			Y						Y	IT
1999	Stratified Architectures	C. Atkinson and T. Kühne		Y	Y	Y	Y	Y			Y	Y					Y				based on AOP	IT
2000	ConcernBASE	Kandé	The ConcernBASE Modeler	Y	Y	Y		Y			Y											IT
2000	Lodwick	F. Steiman	Lodwick	Y	Y							Y			Y							IT, Business
2001	Metapattern	P. Wisse	KnitbITs tool-set for prototyping	Y	Y			Y			Y			Y		Y	Y					Business
2003	SEAM	A. Wegmann	SEAM Cad Tool	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y		Consistency Checking		IT, Business

Advantages: Simple and rich visual notation (r1.8) that allows for modeling the specialization and the composition of roles, relations between roles, assignments of roles to other roles and etc.

3.2. Classification of CBDMs in the context of EA

We begin this section with the CBDM Requirements Checklist Table (see Table 1) where we show how CBDMs from Section 3.1 satisfy requirements for CBDMs. To make a conclusion about the evolution of considered CBDMs, we sort the considered CBDMs by the date of their appearance. We associate this date with a first major publication that we have found in the literature. Note that we also included in this table the Systemic Application column that indicates where each CBDM is more appropriate: in business or IT.

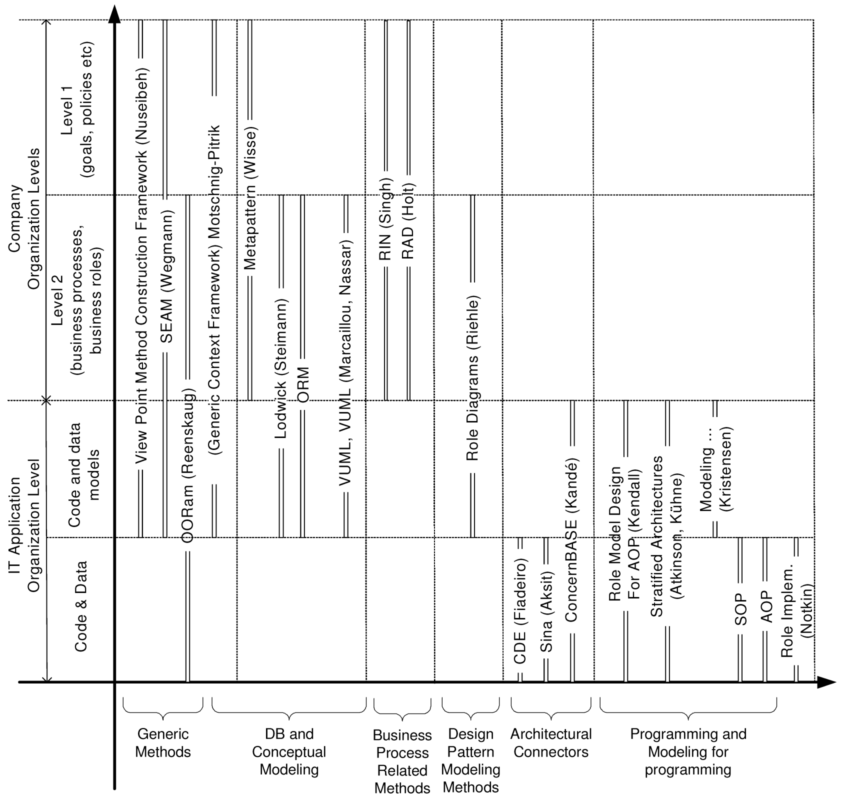
We use Table 1 to associate each method with organization levels where this method can be used. This results in the classification of CBDMs in the context of EA (see Fig. 2, next page). Methods that support multiple levels of EA are more interesting for enterprise architects. Such methods allow an architect to build an abstract picture of an enterprise that makes clear the goals and functionality of the enterprise for its stakeholders.

4. Discussion

In this section we give several observations based on the description of methods from Section 3.1 and the analysis of Table 1 and Fig. 2:

1. Only generic CBDMs can be used for modeling at almost all EA organization levels (from marketing

Fig. 2 The classification of CBDMs in the context of EA (System Application).


Company Organization Levels
Level 1 (goals, policies etc)
Level 2 (business processes, business roles)
IT Application Organization Level
Code and data models
Code & Data
View Point Method Construction Framework (Nuseibeh)
SEAM (Wegmann)
OORam (Reenskaug)
(Generic Context Framework) Motschnig-Pitrik
Metapattern (Wisse)
Lodwick (Steimann)
ORM
VUML, VUML (Marcaillou, Nassar)
RIN (Singh)
RAD (Holt)
Role Diagrams (Riehle)
CDE (Fiadeiro)
Sina (Aksit)
ConcernBASE (Kandé)
Role Model Design For AOP (Kendall)
Stratified Architectures (Atkinson, Kühne)
Modeling ... (Kristensen)
SOP
AOP
Role Implem. (Notkin)
Generic Methods
DB and Conceptual Modeling
Business Process Related Methods
Design Pattern Modeling Methods
Architectural Connectors
Programming and Modeling for programming

to IT implementation). See, for example, OORAM (Reenskaug et al., 1996) or SEAM (Wegmann, 2003). However, often these generic methods cannot be used by all specific specialists in EA team. For example OORAM or SEAM methods are not very appropriate for programmers who work with concerns because these methods are not capable of representing explicitly all the features of concern-based programming languages (like point-cuts in AOP). Therefore EA projects are forced to use several CBDMs that have to be aligned. This alignment can be done by means of choosing a set of similar (or compatible) methods for an EA project or by using a generic method as an alignment tool.

2. CBDMs can be divided roughly into two groups: CBDMs for systems analysis or requirements engineering (methods focusing mostly at company organization levels, see Fig. 2) and CBDMs for code and data design or programming (methods mostly at IT application organization levels, see Fig. 2). These two groups of methods are very different in their objectives. Therefore, this results in a weak integration between methods from these groups. To increase the integration, there is a need for EA methodologies that will help researchers and practitioners to understand and integrate objectives of both groups.
3. Early CBDMs (earlier than the middle of the 90's) aimed mostly to define the main concepts for the representation of system concerns and the semantics of these concepts. Recent-CBDMs, such as OORAM (Reenskaug et al., 1996), SEAM (Wegmann, 2003), Metapattern (Wisse, 2000) and others, aim at building models that can be understood by humans ("human-friendly" models). The problem of making existing approaches more convenient for human reasoning is clearly stated in Chang et al. (1999): "We would like to emphasize informal and yet conceptually precise and practically significant approaches, rather than merely formal languages theory using different formalisms and therefore making them hard to comprehend and compare".
4. Methodological properties of CBDMs are not developed enough: only a few methods support system design verification, consistency checking or model simulation. We understand, however, that the development of methodological properties (especially system design verification) for CBDMs in the context of EA is clearly a difficult task. It requires the integration of different CBDMs (at different organization levels) and the integration of hard (formal techniques) issues with soft (philosophic) issues of modeling.

5. Conclusion

This survey presents the analysis of requirements for concern-based design methods (CBDMs) in the context of enterprise architecture (EA). Based on these requirements, twenty CBDMs were analyzed. This analysis can be used by EA researchers and practitioners to choose appropriate methods for their EA methodologies and to compare CBDMs that they use with similar methods. The important issue (about the requirements for CBDMs) that was not handled in our paper is the question of completeness: Why is the considered set of requirements good enough in the context of EA? In this paper we leave this question open and will investigate it in our future work. The fact that we have used system inquiry as the base of the analysis framework gives us confidence that most of requirements were captured.

We believe that the identified requirements and the analysis method are useful in the context of EA from the "pragmatic" point of view. We hope that our work will help researches to develop and improve their EA methodologies and will bring some order in the mass of concern-based design methods.

References

Aksit M, Tripathi A. Data abstraction mechanisms in SINA/ST, OOPSLA'88, ACM press, San Diego, September 1988;267–275.

Aksit M, Wakita K, Bosch J, Bergmans L, Yonezawa A. Abstracting object interactions using composition filters. *ECOOP'93 Workshop on Object-Based Distributed Programming*, LNCS, Springer-Verlag, 1993;152–184.

Alexander C, Ishikawa S, Silverstein M. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

Andrade L, Fiadeiro J. *Interconnecting Objects Via Contracts*. UML'99, Springer Verlag, 1999;566–583.

Atkinson C, Kühne T, Bunse C. Dimensions of component-based development. *ECOOP'99 Workshop on Component-Oriented Programming* 1999;185–186.

Audi R. *The Cambridge Dictionary of Philosophy*. 2nd ed., Cambridge University Press, 1999.

Barbier F, et al. Formalization of the whole-part relationship in the unified modeling language. *IEEE Transactions on Software Engineering* 2003;29(5):459–470.

Bækdal LK, Kristensen BB. Aggregation from multiple perspectives by roles. *IEEE TOOLS PACIFIC 99*. Melbourne, Australia, 1999;139–150.

Chang SK, et al. The future of visual languages. *IEEE Symposium on Visual Languages*. Tokyo, Japan, 1999;58–63.

Clarke S, Harrison W, Ossher H, Tarr P. Subject-oriented design: Towards improved alignment of requirements, design, and code. *OOPSLA'99* 1999;325–339.

Dori D. *Object-Process Methodology. A Holistic Systems Paradigm*. Heidelberg, New York: Springer Verlag, 2000.

Fiadeiro JL, Lopes A. *Semantics of Architectural Connectors*, TAPSOFT'97, Springer-Verlag, 1997;505–519.

Gamma E, et al. *Design patterns: Elements of Reusable Object-Oriented Software*. Addison-wesley professional computing series, Addison Wesley Publishing Company, 1994.

Gerstla P, Pribbenow S. A conceptual theory of pan-whole relations and its applications. *Data & Knowledge Engineering* 1996;20:305–322.

Harrison W, Ossher H, Subject-oriented programming (a critique of pure objects). *OOPSLA'93*, ACM, 1993;411–428.

Holt AW, Ramsey HR, Grimes JD. Coordination system technology as the basis for a programming environment. *Electrical Communication* 1983;77(4):307–313.

Halpin T. Object role modeling: An overview. *Microsoft Corporation* 2001, retrieved from http://msdn.microsoft.com/library/ on February 16, 2004.

Recommendation X.902, Open Distributed Processing—Basic Reference Model—Part 2: Foundations *ISO/IEC and ITU-T* 1996.

Kandé MM, Strohmeier A. Towards a UML *Profile for Software Architecture. UML'2000*. York, UK, 2000;513–527.

Kendall EA. Goals and roles: The essentials of object oriented business process modeling. *ECOOP'98 Workshop on Object Oriented Business Process Modeling*, 1998.

Kendall EA. Role model designs and implementations with aspect Oriented programming. *OOPSLA'99*. ACM, 1999;353–369.

Kiczales G, et al. Aspect-Oriented Programming. *ECOOP'97*. Springer-Verlag, 1997;220–242.

Kilov H. *Business Specifications: The Key to Successful Software Engineering*. Prentice-Hall, 1999.

Kristensen BB, Object-Oriented Modeling with Roles. *OOIS'95*. Springer, 1995; 57–71.

Lieberherr KJ. Component enhancement: An adaptive reusability mechanism for groups of collaborating classes. *IFIP 12th World Computer Congress on Algorithms, Software, Architecture-Information Processing '92*, 1992;179–185.

Marcaillou S, Kriouile A, Coulette B. VBOOL, une extension d'Eiffel intégrant le concept de point de vue. *MCSEAI'94* 1994; 115–125.

Miller JG. *Living Systems*. University of Colorado Press, 1995.

Motschnig-Pitrik R, Contexts and views in object-oriented languages. *IEEE CONTEXT 99*. LNCS, 1999;1688:256–269.

Motschnig-Pitrik R, Kaasbøll J. Part-whole relationship categories and their application in object-oriented analysis. *IEEE Transactions on Knowledge and Data Engineering* 1999a;11:779–797.

Motschnig-Pitrik. R. Contexts as means to decompose information bases and represent relativized information. *CHI Workshop #11*. Hague. Netherlands, 2000.

Motschnig-Pitrik R. A generic framework for the modeling of contexts and its applications. *Data & Knowledge Engineering*. Elsevier Science Publishers, 2000a;32:145–180.

Motschnig-Pitrik R. The viewpoint abstraction in object-oriented modeling and the uml. *ER'2000*. Salt Lake City, Utah, 2000b;543–557.

Mylopoulos J, Motschnig-Pitrik R, Partitioning information bases with contexts. *CoopIS'95*. Vienna, 1995; 44–54.

Nassar M, et al. Towards a view based unified modeling language. *ICEIS'03*, Angers, France, 2003;257-265.

Nentwich C, Emmerich W, Finkelstein A, Consistency management with repair actions. *IEEE ICSE 03*. Portland, Oregon, 2003;455–464.

Nuseibeh B, Kramer J, Finkelstein A. Expressing the relationships between multiple views in requirements speccification. *ICSE'93* 1993;187–196.

Odeh M, Beeson I, Green S, Sa J. Modeling processes using RAD and UML activity diagrams: An exploratory study. *Arab Conference on Information Technology (ACIT'2002)*, Doha Qatar, 2002.

Ould MA. Business Processes: *Modeling and Analysis for Re-Engineering and Improvement*. John Wiley & Sons, Chichester, 1995.

Ossher H, Tarr P. Multi-dimensional separation of concerns and the hyperspace approach. *Symposium on Software Architectures and Component Technology: The State of the Art in Software Development*. Kluwer, 2001.

Reenskaug T, Wold P, Lehne OA. *Working With Objects: The OOram Software Engineering Method*. Manning Publication Co, 1996.

Rein GL. Collaboration technology for organization design *Hawaii International Conference on System Sciences'93*. Hawaii, 1993; 137–148.

Rescher N, Oppenheim P. Logical analysis of Gestalt concepts. *Brithish Journal for the Philosophy of Science*, 1955;6(22),89–106.

Riehle D, Gross T. Role model based framework design and integration. *OOPSLA'98*, ACM Press, 1998;117–133.

Riehle D. Composite design pattern. *OOPSLA'97*, 1997;218–228.

Riehle D. Describing and composing patterns using role diagrams. *WOON'96*, Russia, St. Petersburg, Electrotechnical University, 1996; 169–178.

Singh B, Rein GL. Role interaction nets (RINs): A process description formalism, technical report CT-083-92. *Microelectronics and Computer Technology Corp*, 1992.

Sowa JF. *Knowledge Representation: Logical, Philosophical, and Computational Foundations, Pacific Grove*, Brooks Cole Publishing Co., 1999.

Steimann F. On the representation of roles in object-oriented and conceptual modeling. *Data and Knowledge Engineering* 2000;35:83–106.

Steimann F, Gößner J, Mück T. On the key role of composition in object-oriented modelling. *UML 2003*, San Francisco, USA, 2003; 106–120

Turner CR, et al. Feature engineering. *IEEE 9th International Workshop on Software Specification and Design*, Ise-Shima (Isobe), Japan, 1998; 162–164.

VanHilst M, Notkin D. Using role components to implement collaboration-based designs. *OOPSLA'96*, San Jose, USA, ACM Press, 1996; 359–369.

Wegmann A. On the systemic enterprise architecture methodology (SEAM). *ICEIS 2003*, Angers, France, 2003; 483–490.

Wisse P. *Metapattern: Context and Time in Information Models*, Addison-Wesley Pub Co; 1st edition, December 15, 2000.

Zachman JA. A framework for information systems architecture. *IBM System Journal* 1987;26(3):276–292.

Pavel Balabko has a Doctorate in the field of enterprise system analysis and design from the EPFL University, Switzerland and a master degree in computer science from Saint-Petersburg Technical University, Russia. He is currently a lead business/system analyst in Luxoft, the global IT outsourcing company with the largest software development and delivery capabilities in Russia. He works in the field of risk management in banking sector.

Alain Wegmann worked for 14 years with Logitech (Switzerland, Taiwan, US) in engineering, manufacturing and marketing functions. He left Logitech in 1997, as VP of Engineering and OEM Marketing, to join the EPFL University, as professor. His research group (http://lamswww.epfl.ch/) develops SEAM: a set of methods and tools for strategic thinking and business/IT alignment. SEAM is based on system thinking and RM-ODP.