A Method for Functional Alignment Verification in Hierarchical Enterprise Models

Irina Rychkova and Alain Wegmann

École Polytechnique Fédérale de Lausanne (EPFL), School of Communication and Computer Science CH-1015 Lausanne, Switzerland
{Irina.Rychkova, Alain.Wegmann}@epfl.ch
http://lamswww.epfl.ch

Abstract. Enterprise modeling involves multiple domains of expertise: requirements engineering, business process modeling, IT development etc. Our experience has shown that hierarchical enterprise models, made of an assembly of system models, are effective. In these models, two hierarchies exist: an organizational level hierarchy (describing systems' construction) and a functional level hierarchy (describing systems' functionality). Using a uniform hierarchical modeling language, system models at different hierarchical levels can be aligned in the context of the enterprise model. Using an operational semantics, each system model can be translated into executable code for model simulation and testing. The possibility to simulate and test models leads to the alignment verification for all system models across both hierarchies.
In this paper we propose a method and tool for functional alignment verification. We use the Abstract State Machine (ASM) and the ASM language (AsmL) to formalize our graphical models for simulation and testing. We illustrate this approach with an example.

1 Introduction

In an IT project, marketing managers, business process designers, IT developers and other specialists develop specific models of an enterprise. Every model highlights properties of the enterprise from a given viewpoint and often requires a specific notation and a modeling tool. As a result, the enterprise models can be seen as a collection of loosely coupled specific models. The main advantage of this modeling approach is that each model is easy to read and understand by the relevant specialist. However it is a challenging task to align the different specific models in the context of one enterprise model.

SEAM [22] stands for a *Systemic Enterprise Architecture Methodology* and represents an enterprise and its environment as an enterprise model. This enterprise model is a set of hierarchical *system models*. To structure these system models (and their relations), SEAM defines a *functional* and an *organizational* hierarchy. The functional hierarchy represents system's behavior at different levels of details. The organizational hierarchy represents systems' construction and related architectural choices.

Alignment across functional and organizational levels requires explicit *refinement relationships* between system models. Refinement relationship has to guarantee the behavioral compatibility of two system models at different hierarchical levels. *Two systems are considered behaviorally compatible if the first system can be replaced by the second one without the environment being able to notice the difference of the system's behavior based on a set of criteria.*(adapted from [18]) The process of checking behavioral compatibility for two system models is the **alignment verification**.

In SEAM we distinguish 2 types of alignment: (1) functional alignment where behavioral compatibility of two system models within one organizational level need to be guaranteed and (2) organizational alignment where behavioral compatibility of two system models at different organizational levels need to be guaranteed. These 2 types of alignment were formalized in [21].

In this paper we present how SEAM system models can be simulated and how functional alignment verification can be achieved. The simulation and verification is based on the Abstract State Machine (ASM)[2] operational semantics. The same method can be extended for organizational alignment verification. This will be addressed in our future work. We illustrate our approach with a model of a Cinema web site.

Section 2 presents the SEAM notation and approach for multi-level model design and alignment verification. Section 3 defines the semantic mapping of SEAM model into ASM executable language (AsmL) for functional alignment verification. Here we also discuss the required tool support. Section 4 presents the related work. Section 5 presents our conclusions.

2 The SEAM Approach for Multi-level Model Design

SEAM is a systemic approach that is applicable to general systems, including IT systems and enterprises. SEAM epistemological principles are based on General System Thinking (GST) [24] and Living Systems Theory (LST) [11]. GST defines system-related concepts such as system boundary, context, etc. LST gives the notion of organizational level. This concept is useful to describe systems that span from technical systems up to companies and markets. SEAM ontology is grounded on the second part of the RM-ODP [18] standard specification. Based on this standard the main modeling concepts such as object, state, action are defined [23]. These concepts are necessary to uniformly and rigorously model systems.

We illustrate our approach with the model of the *Cinema Web Site* shown in fig. 1-2.

Problem specification of the Cinema Web Site. "SEAneMa" is a municipal cinema that develops a web site to provide clients with new services. A Booking tickets service enables tickets reservation via internet. To book tickets a client has to log in on the web site. If logged in, the client can add reservations for a movie of his/her choice to a virtual cart. The movie can be chosen from an agenda - the movie list. The web site has to control a number of places (seats)

available for every movie. Booking tickets finishes when the client commits and logs out.

2.1 An enterprise model at different organizational and functional levels

In this section we present the SEAM ontology and define the concepts of organizational and functional levels in enterprise models.

In SEAM, all entities that have behavior are considered as systems. For example, a value network (group of companies), a company (people and IT systems), IT systems (group of software applications) are all *systems*.

System is modeled as a *working object*. In fig.1.a `Client` is a working object

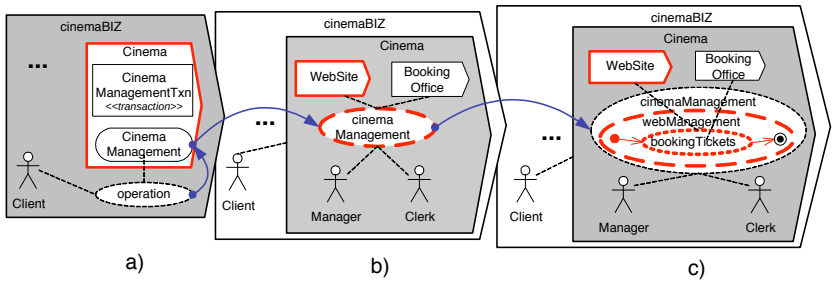

Fig. 1. Model of the `Cinema` at two organizational and functional levels: **a)** `Cinema` (seen as a whole) participates in the business `operation` full interaction (seen as a whole) by doing the `CinemaManagement` partial interaction; **b)** `Cinema` (shown as a composite) performs the `cinemaManagement` full interaction (shown as a whole). The `WebSite` system, the `Manager` human, the `Booking office` system etc define a new organizational level for `Cinema`; **c)** The `cinemaManagement` full interaction shown as a composite. The `webManagement` and `bookingTickets` full interactions define a new functional level for `Cinema`.

that represents a human. `Cinema` is a working object that represents a company.

A working object can participate in a collaboration with other working objects. This collaboration is called a *full interaction* in SEAM. In fig. 1.a the `Client` and the `Cinema` working objects participate in the business `operation` full interaction.

A working object can be decomposed into a set of component working objects. This decomposition defines an **organizational level** of the system. Figure 1.a-b represents the model of `Cinema` at different organizational levels. `WebSite`, `Booking office`, `Manager`, and `Clerk` participating in the `cinemaManagement` full interaction define a new organizational level for `Cinema` and represent its components. A specification of a working object as a whole focuses on its structure and can be considered as a **white box** specification of the system.

A full interaction can be seen as a composition of other full interactions seen as wholes. Interaction seen as a whole defines a **functional level** of the

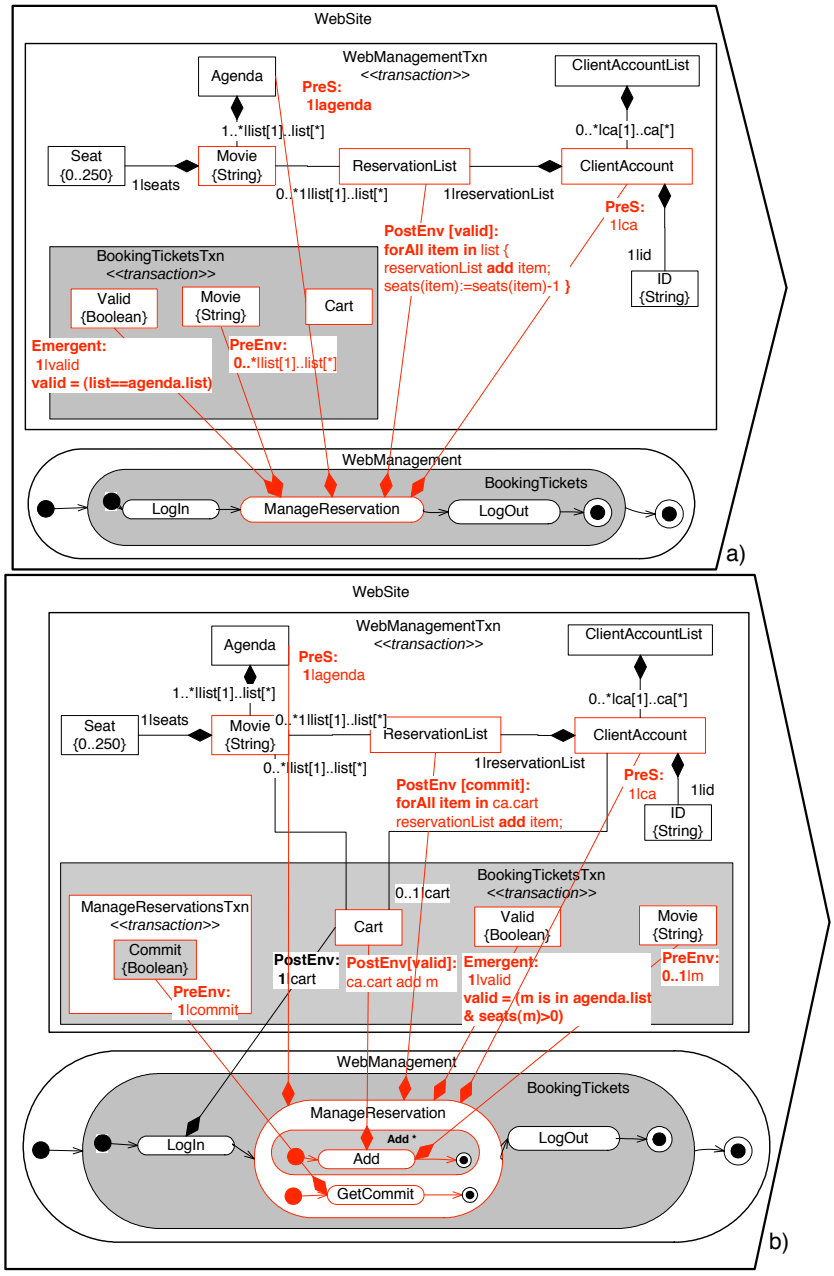

Fig. 2. Model of the cinema `WebSite` at two functional levels: **a)** `WebSite`(seen as a whole) performs `ManageReservation` partial interaction; **b)**`WebSite` (seen as a whole) performs `ManageReservation` partial interaction seen as multiple `Add` reservation partial interactions and one `GetCommit` partial interaction with undefined order. `ManageReservation` finishes when `Commit` parameter received.

system. In fig. 1.b the `cinemaManagement` full interaction is shown as a whole. `CinemaManagement` in fig.1.c is shown as a composite where the `webManagement` full interaction defines a new functional level for `Cinema`.

Working objects and full interactions can have a *participation relation* that shows the participation of the working objects in the full interaction.

A working object represented as a whole is described by its *partial interactions*, *internal actions*, and *properties*. SEAM defines a partial interaction as an action in which the working object communicates with its environment using parameters. An internal action is an action that does not require any exchange with the environment. Properties of a working objects can be instantiated only in the context of an action. Partial interaction and internal action can have a *parameter relation* to a property that indicates that the action access or modify instances of that property. Properties of the working object are encapsulated, i.e. their state can be changed only as a result of a partial interaction or an internal action. A specification of a working object as a whole can be considered as a **black box** specification of the system. Figure 2.a shows the `WebSite` working object as a whole. `Agenda`, `ClientAccountList`, `ClientAccount`, etc are the properties of `WebSite`. The `BookingTickets` partial interaction is shown as a composition of the `LogIn`, `ManageReservation` and `LogOut` partial interactions.

Figure 2.a-b represents the cinema `WebSite` from fig.1.c performing `Booking Tickets` partial interaction at different functional levels. Figure 2.a shows the `ManageReservation` partial interaction as a whole. `ManageReservation` requires an instance of the `ClientAccount` to work with and a list of `Movies` as a parameter, obtained from the environment. This is depicted in the diagram with the parameter relations `PreS, PreEnv` respectively. `ManageReservation` generates the emergent parameter `Valid` that validates if given parameters correspond to the `Movie` list from the `Agenda.` If `valid`, `ManageReservation` places given movies to the client's `ReservationList`. This is depicted with the parameter relation `PostEnv`.

Figure 2.b shows the `ManageReservation` activity: `Add` reservation can be performed multiple times; multiple add reservation (`Add*`) and `GetCommit` partial interactions are performed in any order. This depicted in the diagram using activity transition links. Activity transition links in SEAM specify the execution constraints for full, partial interactions and internal actions (e.g sequentiality, non-determinism, concurrency constraints, etc). Action `Add` requires one `Movie` as a parameter and, if `valid`, places it to the client's virtual `Cart`. It is shown using the parameter relations `PreEnv` and `PostEnv` of the `Add` partial interaction. Instance information is specified on the relation's end. `ManageReservation` finishes when `GetCommit` performed and the `commit` parameter received. `ManageReservation` places `Movies` from the virtual `Cart` to the client's `ReservationList` if `commit` or rolls back if cancelled. Cancel is not shown on the diagram. This figure defines a new functional level for the cinema `WebSite`.

2.2 Operational semantics for the functional alignment verification

To validate and verify the functional alignment of 2 graphical models, we need to check the behavioral compatibility of the corresponding system specifications. To do this an operational semantics for SEAM was defined.

We propose a method for SEAM model simulation based on the transformation of the graphical model into an executable program. To enable such a transformation, we had to check that the SEAM graphical language had all necessary information to simulate the behavior. This did require some improvements to the graphical notation and the addition of stereotypes specific to the simulation.

We also had to select an executable language that supports the appropriate abstraction level to simulate an abstract behavior defined by a SEAM graphical model. In addition, it was important that this language provides an adequate infrastructure for a testing of the executable specification for the alignment verification.

In this work we use the Abstract State Machine (ASM) operational semantics for SEAM to provide the alignment verification for system models. ASM is a method of stepwise refinable abstract operational modeling [2].The choice of ASM as an operational semantics for SEAM was discussed in [19]. The Abstract State Machine language (AsmL) and AsmL environment for model testing (Asmlt) are ASM based tools developed by Microsoft Research group [8], [1]. Using ASM as an operational semantics for SEAM together with AsmL and Asmlt has the following advantages:

Model Simulation support. Any system model at a given functional level can be represented as an asml specification and simulated using AsmL. The modeler can specify a program output and communicate with a program via console.

Model Testing support. Asmlt conformance test analyses the behavioral compatibility of 2 asml specifications. Therefore it can be used for functional alignment verification of system models at different functional levels. For example, to verify that the model in fig. 2.a is behaviorally compatible with the model in fig. 2.b, the conformance test should be performed.

3 The Method and the Tool for Functional Alignment Verification

The ASM operational semantics for SEAM allows interpretation of SEAM modeling concepts in AsmL language and enables the automated generation of executable asml specifications. Based on the definitions of SEAM modeling concepts given in the section 2 and the notion of the AsmL programming concepts [8], we define the AsmL interpretation of SEAM graphical models.

3.1 The AsmL interpretation of SEAM graphical model

We use our Cinema Web Site example and its SEAM model in fig. 1 and fig. 2 to illustrate the interpretation rules for all general modeling concepts in SEAM.

SEAM **working object** is interpreted as an AsmL `namespace`.

---

namespace WebSite *//WebSite working object at fig. 2.a*

SEAM **full interaction** is used for the organizational alignment verification. Can be interpreted as an AsmL `method` that defines the collaboration protocol for asml components representing SEAM working objects.
SEAM **property** can be interpreted as asml types or classes.
- A property encapsulating other property(es) is interpreted as an AsmL `class` with the same name. Encapsulated properties are interpreted as `attributes` of this class. Properties encapsulation is expressed using a specific relation;
- A property encapsulating a collection of other properties of the same type is interpreted as one of the AsmL `instantiated type` (e.g. `Set, Bag, Map`, etc). See Relations for the details;
- A property who's state space is defined as one of the AsmL operational types (e.g. *Integer, Boolean,* etc) is interpreted as an AsmL class with a single attribute that holds a `variable` of this type. If property *type* is *undefined* then the property can be interpreted as a `user declared type`.
Instances of the properties derived from SEAM *parameter relations* can be interpreted as asml `variables`.

---

class ClientAccount *//from the ClientAccount property in fig. 2.a-b*
  var id **as** ID *//from the ClientAccount-ID relation in fig. 2.a-b*
  var reservationList **as** ReservationList? *// ... fig. 2.a-b*
class ID
  var valueRef **as** String *//from the ID property in fig. 2.a-b*
class ReservationList
  var list **as** Bag of Movie *//from the ReservationList property in fig. 2.a-b*
class Movie
  var valueRef **as** String *//from the Movie property in fig. 2.a-b*

SEAM **partial interaction and internal action** are both interpreted as AsmL `methods` with the same name. A *precondition* is interpreted as an AsmL `require assertion`. A *postcondition* is interpreted as AsmL `operation(s)` and/or `ensure assertions`. Both pre- and postconditions are derived from SEAM *parameter relations.*

Add(m **as** Movie, cl **as** ClientAccount) *//from Add partial interaction in fig. 2.b*
  require (m **is in** agenda.list) `&` (seats(m)>0)
  cl.cart.list add m; seats(m):=seats(m)-1;

SEAM **Relation** There are 3 general groups of SEAm relations:
1) Property-Property relations can define attribute-properties for the class-properties or specify AsmL `instantiated types` (e.g. `Set, Bag, Map`, etc);
2) Action-Action relations (activity transitions) define action composition constraints and include plain, conditional, fork, merge transitions, etc. It is interpreted using the AsmL control structures such as `step, if..then..else, forAll, forEach,` etc.;
3) Action-Property relations (parameter relations) are the most important for the simulation. These relations specify action's parameters, pre, post and emergent conditions in terms of property instances. Property instances are interpreted as AsmL `variables`. Action-Property relation holds the instance information: instance name,

state, and state modification instructions (if modification occurs as a result of the action). This information is interpreted as AsmL operation expressions (`:=, new, +, -, include, exclude, ..` etc.). Being combined with the action composition constraints, derived from the Action-Action relations, operation expressions constitutes AsmL method body.

ManageReservation(cl **as** ClientAccount)*//from ManageReservation in fig. 2.b*
  if (commit) **then forAll item in** cl.cart.list
    add item to cl.reservationList.list
    seats(**item**):= seats(item)-1
  . . .

The interpretation rules defined in this section illustrate our approach using a small example of the cinema web site but can be generalized for any SEAM model.

3.2 Simulation tools

The semantic mapping of SEAM modeling concepts into ASM allowed us to develop the SEAM-ASML translator. This tool includes the XML parser, SEAM interpreter, and ASML generator units.

Once a SEAM graphical model is translated into AsmL, it can be simulated to validate if the model reacts correctly on the proposed test cases. Then more formal alignment verification using the Asmlt test environment can be performed. The Asmlt enables different test procedures. The conformance testing is the most interesting in the light of our problem. Here one asml specification (corresponding, for example, to fig.2.a) can be tested against the other one (corresponding to fig.2.b) to check their behavioral compatibility. Positive result of this test informs the modeler that asml specifications are behaviorally equivalent and, respectively, means that graphical models at different functional levels are functionally aligned.

SeamCAD is a web-based Computer Aided Design (CAD) tool[10],[9] that allows drawing and storing of SEAM hierarchical models. The interoperability of SeamCAD with the SEAM-ASML translator is supported by using an XML intermediate format for the graphical specifications.

4 Related Work

Many languages for hierarchical modeling exist: Conceptual Graph[16], Catalysis[6], TROPOS[12], UML[20], DEMO[3], OPM[4], BPEL[14], etc.

TROPOS [12] is a requirements engineering method that aligns its specifications by providing a goal refinement technique. This method does not consider behavioral equivalence for model alignment.

Catalysis [6] proposes hierarchical modeling (IT systems, components and programming classes) and aligns its models using a top-down design. The modeling principles of Catalysis are based on UML. This method does not propose a formal semantics for its models apart from semantics exist for UML.

DEMO[3] is an EA framework originated from the organizational theory called Language/Action Perspective. DEMO defines its organizational levels based on a communication paradigm. Functional levels are defined in DEMO based on the view of

business processes as transactions. DEMO provides an operational semantics for model formalization. However alignment verification is not defined in this method.

OPM[4],[5] proposes a method for the complete integration of the systems' states and behaviors within a single graphical model. OPM provides a visual notation and defines an operational semantics for model simulation. It does not provide model checking (i.e. alignment verification).

BPMN[15] and BPEL[14] provide a visual notation and a formalism for business process model development, simulation, and verification. Operational semantics for BPEL was defined using Abstract State Mashine (ASM)[7].

The semantics of activity diagrams in UML 2.0 is based on Petri Nets[17]. However, there were many attempts to define semantics of activity diagrams based on other formal languages: LOTOS, ASM, CSP, LTS (see [17] for details).

SEAM method focused on the parallel design across organizational and functional hierarchies. It covers different domains and defines the solution for the functional alignment verification based on the principle of the behavioral compatibility [18], [13]. SEAM uses the ASM operational semantics for model simulation and verification.

5 Conclusion

In this paper we defined the concepts of organizational and functional alignment and proposed the functional alignment verification technique. We presented SEAM method for hierarchical system development and specified its modeling concepts. Since the transition from the descriptive graphical model to the prescriptive executable program is not straightforward, graphical language as well as executable language have to satisfy certain criteria (i.e. possibility to specify the operational data for a graphical language and abstraction level support for executable language). We use ASM and its executable language AsmL as an operational semantics for SEAM. At the last section the AsmL interpretation of SEAM modeling concepts was presented. Based on this interpretation SEAM-ASML translator tool was developed. Using SEAM-ASML translator, one can obtain an executable asml specification of a system at any given organizational and functional level. Functional alignment verification is performed using Asmlt conformance testing.

References

1. Barnett, M., Grieskamp, W., Nachmanson, L., Schulte, W., Tillmann, N., Veanes, M.: Towards a Tool Environment for Model-Based Testing with AsmL.
2. Börger, E., Stärk, R.: Abstract State Machines. A Method for High-Level System Design and Analysis. Springer-Verlag, Berlin Heidelberg New York (2003)
3. Dietz, J. L. G.: DEMO: towards a discipline of Organisation Engineering. 1999.
4. Dori, D., Object-Process Methodology, A Holistic Systems Paradigm. 2002: Springer Verlag.
5. Dori, D., Reinhartz-Beger, I., Sturm, A. OPCAT - A Bimodal CASE Tool for Object-Process Based System Development. in ICEIS 2003. 2003. Angers, France.
6. D'souza, F.D., Wills, C. A.: Object, Components and Frameworks with UML, The Catalysis Approach. 1999: Addison-Wesley.
7. Farahbod R., Glsser U., Vajihollahi M.: Abstract Operational Semantics of the Business Process Execution Language for Web Services, Simon Fraser University, Tech. Report #SFU-CMPT-TR 2004-03, 2004

8. http://research.microsoft.com/fse/asml/
9. Lê, L.S., Wegmann, A.: Definition of an Object-Oriented Modeling Language for Enterprise Architecture. to be published in HICSS 2005. Hawaii, USA.
10. Lê, L.S., Wegmann, A.: SeamCAD 1.x: User's Guide, School of Computer and Communication Sciences, EPFL, Lausanne Switzerland, Report No. IC/2004/98, November 2004.
11. Miller, J.G.: Living Systems. University of Colorado Press, 1995.
12. Mylopoulos, J., Kolp, M., Castro, J.: UML for Agent-Oriented Software Development: The Tropos Proposal, Proceedings of the 4th international conference on the Unified Modeling Language UML 2001, Toronto, Canada, October 1-5, 2001.
13. Philippi, S.: Formally based modeling and inheritance of behaviour in object-oriented systems. Journal of Systems and Software, Feb 2004.
14. Specification: Business Process Execution Language for Web Services Version 1.1, The IBM, 2004.
15. Specification: Business Process Modeling Notation (BPMN) Version 1.0, Business Process Management Initiative (BPMI),May 3, 2004
16. Sowa, J.,F.: Knowledge Representation: Logical, Philosophical, and Computational Foundations. Pacific Grove, Brooks Cole Publishing Co., 1999
17. Störrle, H.: Semantics of UML 2.0 Activities, Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '04), Rome, Italy, September 26-29, 2004.
18. Reference model of open distributed processing part 1. Draft International Standard (DIS), Helsinki, Finland, (15-18 May 1995)
19. Rychkova, I., Wegmann, A., Balabko, P.: Operational ASM Semantics behind Graphical SEAM Notation. Workshop DAIS-FMOODS'03, Paris 2003.
20. Unified Modeling Language: Superstructure (final adopted spec, version 2.0)", Technical report, Object Management Group, November 2003.
21. Wegmann, A., Balabko, P., Lê, L.S., Regev, G., Rychkova, I.: A Method and Tool for Business-IT Alignment in Enterprise Architecture. CAiSE Forum'05, Porto, Portugal
22. Wegmann, A.: On the systemic enterprise architecture methodology (SEAM). Published at the International Conference on Enterprise Information Systems 2003 (ICEIS 2003), Angers, France.
23. Wegmann, A., Naumenko, A.: Conceptual Modeling of Complex Systems Using an RM-ODP Based Ontology. Proceedings of the 5th IEEE International Enterprise Distributed Object Computing Conference - EDOC 2001, Seattle, USA, September 2001, pp. 200-211.
24. Weinberg, G. M.: An Introduction to General Systems Thinking. New York: Wiley & Sons, 1975.