Knowledge-Based Systems 31 (2012) 89–105

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Formal framework to support organizational design ☆

Catholijn M. Jonker [a], Viara Popova [b], Alexei Sharpanskykh [c,*], Jan Treur [c], Pınar Yolum [d]

[a] Interactive Intelligence group, Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD Delft, The Netherlands
[b] Institute of Computer Science, University of Tartu, J. Liivi 2, Tartu 50409, Estonia
[c] Agent Systems Research Group, VU University Amsterdam, De Boelelaan 1081a, NL-1081 HV Amsterdam, The Netherlands
[d] Department of Computer Engineering, Bogazici University, TR-34342 Bebek, Istanbul, Turkey

ARTICLE INFO

Article history:
Received 19 July 2011
Received in revised form 10 January 2012
Accepted 17 February 2012
Available online 28 February 2012

Keywords:
Organization design
Organizational structure
Organizational dynamics
Design tools
Design operators
Ontologies

ABSTRACT

Organizational design is an important topic in the literature on organizations. Usually the design principles are addressed informally in this literature. This paper makes a first attempt to formally introduce design operators to formalize the design steps in the process of designing organizations. These operators help an organization designer create an organization design from scratch as well as offer the possibility to revise existing designs of organizations. The operators offer both top-down refinements and bottom-up grouping options. Importantly, the operators can be combined into complex operators that can serve as patterns for larger steps in an organization design process. The usability of the design operators is demonstrated in a running example. The contribution of this paper provides a solid basis for the development of a software environment supporting interactive organization design processes. This is demonstrated by an implemented prototype example tool.

1. Introduction

Organizations play a key role in the modern society. To a large extent, the vitality and productivity of an organization situated in an environment of a certain type depend on the kinds of structure and behavior of the organization that should conform to the environmental conditions. Business modeling is instrumental in addressing a number of problems such as: understanding the structure and dynamics of the organization [24], diagnosing problems and detecting avenues for improvement [15], ensuring common vocabulary and understanding, and formulating requirements needed for interoperation. A number of frameworks and tools for business modeling have been developed such as the OMG Business Modeling and Management Specifications (http://www.omg.org), ARIS [37], CIMOSA [14], MEMO [17], and IBM WebSphere Business Modeler. Less comprehensive with respect to scope, but still relevant, is the research on Enterprise Ontology [13] which aims at defining an abstract high-level model that captures the essence of the organization and thus enabling easier communication and shared understanding between inter- and intra-organizational parties.

The Business architecture defines the structure of the enterprise including its business processes and information and governance structure and is typically organized in a number of views clustering related aspects of the organizational structure and dynamics. For example, one important view is the organization-oriented view which defines the structure of the organization into business units, roles and their relationships, communication and capabilities. Another essential view is the process-oriented view which describes the business processes and tasks, the material and information resources needed for and produced by these processes and so on. This view has been the focus of extensive research and can be modeled by a number of existing methods and tools such as PetriNets [46], BPMN (http://www.bpmn.org/), EPC [47], and YAWL (http://www.yawlfoundation.org/). Some authors consider communication relations between organizational actors as organizational processes too (see e.g., [16]).

In this paper, for reasons of specificity, we focus on the organization-oriented view in the context of the framework presented in [31–34,41,42,21]. It is beyond the scope of the paper to discuss how the chosen framework compares to other existing ones—for this the reader is referred to [41]. The approach presented here, however, can be extended and adapted to other existing frameworks as well.

The specific problem addressed in the paper is to formalize the process of organizational design and organizational change in the context of the organization-oriented view.

Organization design is concerned "with what an organization is ought to be" [30]. More specifically, Galbraith [18] stated that orga-

☆ A preliminary (short) version of this manuscript appeared in Proceedings of the Second International Conference on Design Computing and Cognition (DCC'06).

* Corresponding author. Address: Department of Computer Science, Faculty of Science, VU University Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands. Tel.: +31 205985887; fax: +31 205987653.

E-mail address: sharp@few.vu.nl (A. Sharpanskykh).

0950-7051/$ - see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2012.02.011

nization design "is conceived to be a decision process to bring about a coherence between the goals or purposes for which the organization exists, the patterns of division of labor and interunit coordination and the people who will do the work". Further Galbaith argues that design is an essential process for "creating organizations, which perform better than those, which arise naturally".

In the literature, a range of theories and guidelines concerning the design of organizations are present [18,12,28,5]. For example, Duncan proposed a contingency model for designing organizations with environmental variables being the principal determinants of organizational models. Mintzberg described a number of guidelines applicable mostly to designing hierarchical organizations that function in a relatively stable environment. However, despite the abundance of organizational design theories no general principles applicable to organizational design at all times and places can be identified [39]. Moreover, almost all theoretical findings in organizational design are informal and often vague. In order to provide an organization designer or a manager with operational automated tools for creating, analyzing, and revising organizations, in the first place a formal representation of an organization model as a design object description should be provided. In addition to this, to address the operations performed on such design object descriptions during a design process, a formal representation of design operators underlying possible design steps is needed. Such design operators describe the possible transitions between design object descriptions. Using the design operators, a design process can be described by choosing, at various points in time, the next operator to be applied to transform the current design object description into the next one. Examples of very simple design operators are adding or deleting an element of a design object description. More sophisticated design operators can involve, for example, the introduction of further refinement of the aggregation levels within a design object description. In this paper we introduce a formal organizational model format, to be used to represent design object descriptions for the organization-oriented view. On top of this, a set of design operators is formally defined. The formalization is based on the sorted predicate logic [26].

Often in the literature organizational design is recognized as an engineering problem [10]. From this perspective, design is considered as a continuous process of a gradual change of an organizational model by applying certain operations [30]. For example, [28] describes the design process as the following sequence of operations: given overall organizational needs, a designer refines the needs into specific tasks, which are further combined into positions. The next step is to build the "superstructure" by performing unit grouping using special guidelines and heuristics (e.g., grouping by knowledge and skill, by work process and function, by time, by place, etc.). Then, the grouping process is repeated recursively, until the organization hierarchy is complete.

For this paper, we aimed at identifying the most commonly and generally used set of operators for designing organizations. For this purpose, the literature from social sciences, and design principles used in other disciplines were investigated. For example, useful principles for organizational design can be found in the area of derivative grammars. Thus, graphical changes in organizational designs may be described by shape [45] and graph grammars [36]. Whereas changes in textual (or symbolic) structural and dynamic descriptions of organizational elements may be specified by string [11] and graph grammars, which allow representation of relationships between the descriptions of different elements. In order to relate graphical organizational designs to designs described in a symbolic form, parallel grammars (or grammars defined in multiple algebras) may be used [45]. For designing organization structures with multiple levels of representation (e.g., hierarchical organizations with departments, groups, sections) abstraction grammars [38] and hierarchical graph grammars [19] can be useful. By means of abstraction grammars, design is performed from the top level of the abstraction hierarchy to the bottom (most concrete) level, with each design generation using the prior level design as a pattern. Furthermore, mechanisms for choosing the most appropriate design generated by different transformations defined by grammars have been developed in different areas (e.g. recursive annealing in mechanical design [38]). Although it is widely recognized in social studies that no "best" design of an organization exists, a number of informal guidelines and best practices developed in the area of organizational design can help in identifying the most suitable organizational designs.

Thus, based on the rich literature on design, this paper makes a first attempt to formalize the operators underlying organization design processes. A set of design operators is formally introduced, which provides the means for creating a design of an organization from scratch as well as revising existing designs for organizations. Furthermore, the formalization of the operators provides a solid basis for a software tool supporting interactive organization design processes. A formal organizational specification provides a clear overview of organizational structures and dynamics, which would facilitate decision making by organizational managers. Furthermore, a formal, consistent specification of an organization may be used for analysis of organizational structures and interactions (e.g., identifying inconsistencies and bottlenecks in an organizational structure) by managers as a part of a dedicated knowledge-based system. Such a system may be also used for organizational design:

– it may advise on the choice of design operators based on the type of the organization under consideration (e.g., hierarchical, flat organic);
– it may be able to trace the design operations of the user and to suggest suitable design operators based on design patterns stored in the system.

The research presented in this paper does not overlap with the area of organizational learning [1,2,40,29], however there are some meeting points between the two areas. More specifically, within the technical view on organizational learning, the two areas can meet when organizational learning results in organizational change that will be reflected in the formal specification structure of the organization and/or the roles of the organization. Our research concerns the process of incorporating new elements and properties in the specification. It does not address the question of what these elements and properties should be in order to achieve a more faithful representation or to improve the functioning of the organization, nor does it consider the process of deriving them.

In Section 2 a formal framework for the specification of design object descriptions for organizations is described. Sections 3 and 4 introduce a set of classes of operators to create and modify design object descriptions for organizations. In Section 5 checking consistency of organizational specifications during the organizational design is considered. Section 6 illustrates the application of a developed prototype by an example. Finally, Section 7 discusses future work and provides general conclusions.

2. Format for an organizational model as a design object description

We consider a generic organization model, abstracted from the specific instances of agents (actors), which consists only of structural and behavioral descriptions of organizational roles and relations between them. A top-down ordering of definitions is used, meaning that concepts are referred to before they are defined.

Definition 1 (*A specification of an organization*). A specification of an organization with the name O is described by the relation is_org_described_by (O, Γ, Δ), where Γ is a structural description and Δ is a description of dynamics.

An organizational structure is characterized by the patterns of relationships or activities in an organization, and described by sets of roles, groups, interaction and interlevel links, relations between them and an environment.

Definition 2 (*A structural description of an organizational specification*). A structural description Γ of an organizational specification described by the relation is_org_described_by (O, Γ, Δ) is determined by a set of relations, among which[1]:

- a relation has_basic_components (Γ, R, G, IL, ILL, ONT, M, ENV) defined on the subsets R, G, IL, ILL, ONT, M, ENV of the corresponding general sets ROLE (the set of all possible role names), GROUP (the set of all possible group names), INTERACTION_LINK (the set of all possible interaction links names), INTERLEVEL_LINK (the set of all possible interlevel links names), ONTOLOGY (the set of all possible ontology names), ONTO_MAPPING (the set of all possible ontology mappings names), ENVIRONMENT (the set of all possible environment names)[2]
- a relation for specifying a role $r \in R$ in Γ is_role_in (r, Γ)
- a relation for specifying an interaction link $e \in IL$ in Γ is_interaction_link_in (e, Γ)
- a relation for specifying an interlevel link $il \in ILL$ in Γ is_interlevel_link_in (il, Γ)
- a relation for specifying an environment $env \in ENV$ is_environment_in (env, ENV)
- a relation has_input_ontology (r, o) that assigns an input ontology $o \in ONT$ to a role $r \in R$ (similarly the relations for output, internal, and interaction ontologies are introduced: has_output_ontology (r, o), has_interaction_ontology (r, o), has_internal_ontology (r, o))
- a relation has_input_ontology (env, o) that assigns an input ontology $o \in ONT$ to an environment $env \in ENV$ (similarly the relations for output, internal, and interaction ontologies are introduced: has_output_ontology (env, o), has_interaction_ontology (env, o), has_internal_ontology (env, o))
- a relation is_ontology_for (el, o) that assigns an ontology $o \in ONT$ either to a role $el \in R$ or an environment $el \in ENV$
- a relation has_onto_mapping (il, m) that associates an interlevel link $il \in ILL$ with an ontology mapping $m \in M$ (an ontology mapping for an interaction link is defined similarly)
- a relation is_interaction_link_of_type (e, type) that specifies an interaction link $e \in IL$ of one of the types: role_interaction_link, env_input_link, env_output_link
- a relation connects_to (e, r, r′, Γ) that specifies a connection by an interaction link $e \in IL$ from a source-role $r \in R$ to a destination role $r' \in R$ in Γ
- a relation connects_to (e, env, r, Γ) that specifies a connection by an interaction link $e \in IL$ of type env_output_link from an environment $env \in ENV$ to a role $r \in R$ in Γ (similarly for connects_to (e, r, env, Γ))
- a relation subrole_of_in (r′, r, Γ) that specifies a subrole $r' \in R$ of a role $r \in R$ in Γ
- a relation member_of_in (r, g, Γ) that specifies a member role $r \in R$ of a group $g \in G$ in Γ
- a relation interlevel_connection (il, r, r′, Γ) that specifies a connection by an interlevel link $il \in ILL$ between roles $r, r' \in R$ of adjacent aggregation levels (i.e., between a role and one of its subroles)

Organizational behavior is described by dynamic properties of the organizational structure elements.

Definition 3 (*A description of dynamics of an organizational specification*). A description of dynamics Δ of an organizational specification described by the relation is_org_described_by (O, Γ, Δ) is determined by a set of relations, among which:

- a relation has_basic_components (Δ, DP) that specifies a set of dynamic properties names DP defined in an organizational specification
- a relation has_dynamic_property (r, d) that specifies a dynamic property $d \in DP$ for a role $r \in R$ (the relations for dynamic properties of an interlevel link, a group and an environment are defined in a similar manner: has_dynamic_property (e, d), has_dynamic_property (g, d), has_dynamic_property (env, d))
- a relation has_expression (d, expr) that identifies a dynamic property name $d \in DP$ with a dynamic property expression $expr \in DPEXPR$ (e.g., a formula in sorted first-order predicate logic)

A role is a basic structural element of an organization. It represents a subset of functionalities, performed by an organization, abstracted from specific agents (or actors) who fulfill them. Each role has an input and an output interface, which facilitate the interaction (communication) with other roles. The interfaces are described in terms of interaction (input and output) ontologies: a vocabulary or a signature specified in order-sorted logic. An ontology contains objects that are typed with sorts, relations, and functions. Generally speaking, an input ontology determines what types of information are allowed to be transferred to the input of a role, and an output ontology predefines what kinds of information can be generated at the output of a role.

Each role can be composed of a number of other roles, until the necessary detailed level of aggregation is achieved. Thus, roles can be specified and analyzed at different aggregation levels, which correspond to different levels of an organizational structure. A role that is composed of (interacting) subroles, is called a composite role.

Definition 4 (*Role*). A specification of a role r is determined by:

Objects:

- or, oi, o, o′, o″ $\in$ ONT, or = o $\cup$ o′ $\cup$ o″, oi = o′ $\cup$ o″, here $\cup$ is a functional symbol that maps names of ontologies to a name of the joint ontology

Relations:

- has_internal_ontology (r, o), has_input_ontology (r, o′), and has_output_ontology (r, o″)
- has_ontology (r, or) and has_interaction_ontology (r, oi)
- $d \in DP$, has_dynamic_property (r, d)

The ontologies, which describe interfaces of interacting roles, can be different. Therefore, if necessary, the specification of a role interaction process includes ontology mapping. An ontology mapping m between ontologies o and o′ is characterized by a set of relations is_part_of_onto_map (a, a′, m), where a is an atom expressed in ontology o and a′ is an atom expressed using ontology o′.

Definition 5 (*Ontology mapping*). An ontology mapping m between ontologies o and o′ is characterized by:

[1] Notice that all the following relations are defined using the names of organization elements; the specifications for these elements will be provided in the following definitions.

[2] The difference between R and ROLE, for example, is that R (subset of ROLE) is the set of all role names that occur in Γ.

- is_part_of_onto_map (a, a′, m), where $a \in At(o)$ and $a' \in At(o')$
- for $a \in At(o)$ is_in_domain_of $(a, m) \Leftrightarrow \exists a' \in At(o')$ is_part_of_onto_map (a, a′, m), where At (o) is the set of all atoms, expressed in ontology o.
- for $a' \in At(o')$ is_in_range_of $(a', m) \Leftrightarrow \exists a \in At(o)$ is_part_of_onto_map (a, a′, m)

Roles of the same aggregation level interact with each other by means of interaction links. The interaction between roles is restricted to communication acts.

Definition 6 (*Interaction link*). An interaction link e is determined by:

Relations:

- is_interaction_link_in (e, Γ)
- has_onto_mapping (e, m) for some $m \in M$
- has_dynamic_property (e, d) for a number of $d \in DP$

Constraints:

- An interaction link e must connect two roles at the same aggregation level: is_interaction_link_in (e, Γ) $\Rightarrow \exists r, r' \in R$ connects_to (e, r, r′, Γ)$\wedge \neg$has_subrole (r, r′)$\wedge \neg$has_subrole (r′, r)

An interlevel link connects a composite role with one of its subroles. It represents an information transition between two adjacent aggregation levels. For roles connected by an interlevel link, this link is described by an ontology mapping between the corresponding elements of ontologies, part of which may be identity correspondence. Moreover, an ontology mapping associated with an interlevel link may be used for representing mechanisms of information abstraction. These mechanisms can be applied for transmitting (or generating) partial, aggregated or generalized information to the input (or from the output) of a role.

Definition 7 (*Interlevel link*). A specification for an interlevel link il is determined by:

Relations:

- is_interlevel_link_in (il, Γ)
- has_onto_mapping (il, m) for some $m \in M$

Constraints:

- An interlevel link il must connect two roles at two adjacent aggregation levels: is_interlevel_link_in (il, Γ) $\Rightarrow \exists r, r' \in R$ subrole_of_in (r′, r, Γ)$\wedge$ (interlevel_connection (il, r, r′, Γ)$\vee$ interlevel_connection (il, r′, r, Γ))

A group is a composite structural element of an organization that consists of a number of roles. In contrast to roles, a group does not have well-defined input and output interfaces. Groups can be used for modeling units of organic organizations, which are characterized by loosely defined or sometimes informal frequently changing structures that operate in a dynamic environment. Furthermore, groups can be used at the intermediate design steps for identifying a collection of roles, which may be further transformed into a composite role.

Definition 8 (*Group*). A group g is defined by the relations to other concepts:

- membership relation member_of_in: $r \in R$ member_of_in (r, g, Γ)
- has_dynamic_property (g, d) with $d \in DP$

The conceptualized environment represents a special component of an organization model. According to some sociological theories (e.g., contingency theory), an environment represents a key determinant in organizational design, upon which an organizational model is contingent. Similarly to roles, the environment is represented in this proposal by an element having input and output interfaces, which facilitate in interaction with roles of an organization. The interfaces are conceptualized by the environment interaction (input and output) ontologies. Interaction links between roles and the environment are indicated in the organizational model as links that have a specific type, namely env_input_link or env_output_link by means of the predicate is_interaction_link_ of_type.

The internal structure of the environment is not fixed, i.e., the designer has freedom to provide his/her own conceptualization of the environment. For example, the environment can be defined by a set of objects with certain properties and states and by causal relations between objects. On the one hand, roles are capable of observing states and properties of objects in the environment; on the other hand, they can act or react and, thus, affect the environment. We distinguish passive and active observation processes. For example, when some object is observable by a role and the role continuously keeps track of its state, changing its internal representation of the object if necessary, passive observation occurs. For passive observation, no initiative of a role is needed. Active observation is always concerned with the role's initiative and focusing. For particular purposes the internal specification for the environment can be conceptualized using one of the existing world ontologies (e.g., CYC, SUMO, TOVE [4]). However, despite the richness and the extensiveness of these ontological bases, more specific and refined types of concepts and relations are required for modelling particular types of organizations and environments.

Definition 9 (*Environment*). A specification of an environment env is determined by:

Objects:

- oe, oi, o, o′, o″ $\in$ ONT, $oe = o \cup o' \cup o''$ and $oi = o' \cup o''$

Relations:

- has_internal_ontology (env, o), has_input_ontology (env, o′), and has_output_ontology (env, o″)
- has_ontology (env, oe) and has_interaction_ontology (env, oi)
- $d \in DP$, has_dynamic_property (env, d)

Constraints:

- $IL' \subseteq IL$, $\forall e \in IL'$ is_interaction_link_in (e, Γ) $\Rightarrow \exists r' \in R$ such that connects_to (e, env, r′, Γ) $\vee$ $\exists r'' \in R$ such that connects_to (e, r″, env, Γ)

The behavior of each element of an organizational structure is described by a set of dynamic properties. With each name of a dynamic property, an expression is associated. Dynamic property expressions represent formulae specified over a certain ontology (ies). In particular, a dynamic property for a role is expressed using a role ontology. A dynamic property for an interaction link is constructed using the output ontology of a role-source of a link and the input ontology of a role-destination. A group dynamic property is expressed using ontologies of roles- members of a group.

Definition 10 (*Dynamic Property*). A specification of a dynamic property $d \in DP$ is described by:

- has_expression (d, expr) for some $expr \in DPEXPR$
- uses_ont (d, o) for some $o \in ONT$

- if $r \in R$ and has_dynamic_property (r, d), then uses_ont (d, o) ⇒ has_ontology (r, o)
- if $e \in IL$ and has_dynamic_property (e, d), then uses_ont (d, o) ⇒∃r, $r' \in R$, ∃o′, o″ ∈ ONT connects_to (e, r, r′, Γ)∧ has_output_ontology (r, o′)∧ has_input_ontology (r′, o″) ∧ $o \subseteq o' \cup o''$
- if $g \in G$ and has_dynamic_property (g, d), then uses_ont (d, o) ⇒∃r ∈ R member_of_in (r, G, Γ)∧ has_ontology (r, o)

Dynamic properties expressions are specified in the Temporal Trace Language (TTL) [22,43,44], which is a variant of order-sorted predicate logic [26]. TTL is a hybrid language; it combines the expressivity of logic-based (automata-based) languages [3] with the numerical expressivity of Dynamical Systems Theory [35] based on differential equations. In particular, it allows:

– discrete and continuous temporal modelling of a system at different aggregation levels;
– numerical expressivity for modelling systems with explicitly defined quantitative relations best presented by difference or differential equations;
– specifying qualitative aspects of a system by expressing logical relationships between parts of a system.

Furthermore, using dedicated tools based on TTL [8] both the generation and formalization of simulated and empirical trajectories or traces, as well as analysis of complex dynamic properties of such traces and relationships between such properties can be performed.

To enable reasoning about the dynamic properties the language TTL includes special sorts, such as: TIME (a set of linearly ordered time points), STATE (a set of all state names of a system), and TRACE (a set of all trace names; a trace or a trajectory can be thought of as a timeline with a state for each time point).

Definition 11 (*Dynamic Property Expression*). Dynamic Property Expression is constructed as follows:

1. STATOM ⊆ ONT and has_expression: STATOM × STATOMEXPR whereSTATOM denotes a set of static atoms in an ontology.
2. Static property expressions (STATPROPEXPR) are generated by applying conjunction, disjunction, implication, and negation operators on STATOMEXPR and STATPROPEXPR.
3. States relate to particular time points in traces (TRACE × TIME → STATE). States are related to state properties via the satisfaction relation |=, formally defined as a binary infix predicate (or by holds as a binary prefix predicate). For example, the expression state (γ: TRACE, t: TIME, output (r: ROLE)) | = p (or holds (state (γ, t, output (r)), p)) denotes that state property p holds in trace γ at time t in the output state of role r.
4. The set of all dynamic properties expressions (DPEXPR) for the corresponding dynamic properties names (DP) is inductively defined by:
   (1) If v_1 is a term of sort STATE, and u_1 is a term of the sort STATPROPEXPR, then holds (v_1, u_1) is an atomic dynamic property expression (belongs to the sort DPATOMEXPR, which is a subsort of the sort DPEXPR).
   (2) If τ_1, τ_2 are terms of any TTL sort, then $\tau_1 = \tau_2$ is an atomic dynamic property expression.
   (3) If t_1, t_2 are terms of sort TIME, then $t_1 < t_2$ is an atomic dynamic property expression.
   (4) The set of dynamic properties expressions (sort DPEXPR) is defined inductively based on atomic dynamic property expressions using boolean propositional connectives and quantifiers (∧, ∨, ⇒, ¬, ∃, ∀).

The application of the basic components of an organizational model is illustrated by means of a running example. Consider the process of organizing a conference. A partial model for the considered conference organization is shown in Fig. 1.

At the most abstract level 0 the organization is specified by one role CO (Conference Organization) that interacts with the environment Env. Role CO can act in the environment, for example by posting a call for papers in different media. Note, that the organizational model is depicted in a modular way; i.e., components of every aggregation level can be visualized and analyzed both separately and in relation to each other. Consequently, scalability of graphical representation of an organizational model is achieved. At the first aggregation level the internal structure of the composite role CO is revealed. It consists of subrole Ch (Conference Chair), which interacts with two other subroles: OC (Organizing Committee) and PS (the Paper Selection role). At the second aggregation level the internal structure of role PS is represented. It consists of subrole PCh (Program Chair), subrole PCM (Program Committee Member), and subrole R (Reviewer), which interact with each other. The input interface of role PS is connected to the input interface of its subrole PCh by means of an interlevel link. In our example the interlevel link describes the mapping between the input ontology of role PS and the input ontology of its subrole PCh. It means that information, transmitted to the role PS at the first aggregation level, will immediately appear at the input interface of subrole PCh, expressed in terms of its input ontology at the second aggregation level.

For example, if Ch requests some information from PS, the request actually arrives at the input of PCh. As a result of the internal communications among PCh, PCM and R, PCh will generate a reply that will appear as a response of PS for Ch.

For each element of the considered organizational model a set of dynamic properties is identified and formally specified in TTL. In fact, these properties define constrains on the behavior of elements, thus forming their expected behavioral repertoire in the organization.

For example, for the role Reviewer the dynamic property may be specified expressing that a reviewer should send his/her review to the Program Chair before a certain deadline. This property is expressed in TTL as follows:

∀t state (γ, t, environment) |= deadline_for_conference (d) ⇒ ∃t′ < d state (γ, t′, output (Reviewer)) |= communicate_from_to (Reviewer, Program_Chair, inform, review_report)

The predicate communicate_from_to (r1:ROLE, r2:ROLE, s_act:SPEECH_ACT, message:STRING) is used to specify the speech act s_act (e.g., inform, request, ask) from role-source r1 to role-destination r2 with the content message.

3. Representing design operators for organizational design

In this section, a formal format for representing design operators is presented and, based on this format, formulations are introduced for a number of primitive design operators for designing organizations. Each primitive operator represents a specialized one-step operator to transform a design object description (organizational model) into a next one. Each operator is concerned with a part of the design object description to which it will be applied and the part of the transformed design object description, resulting from the operator application. The parts of the organization O that are being modified in terms of structure and dynamics (i.e., sets of dynamic properties) are specified using the in-focus relations: structure_in_focus (O, Rf, Gf, ILf, ILLf, ONTf, Mf, ENVf) and dynamics_in_focus (O, DPf), with Rf ⊆ R, Gf ⊆ G, ILf ⊆ IL, ILLf ⊆ ILL, ONTf ⊆ ONT, Mf ⊆ M, ENVf ⊆ ENV, DPf ⊆ DP. The remaining parts of the organization stay the same.

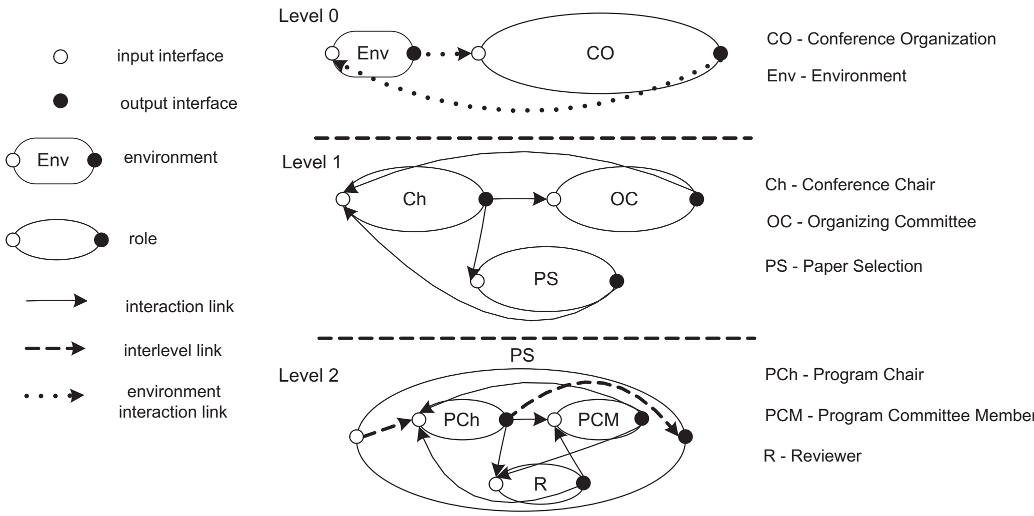


Fig. 1. Model of the conference organizing committee.

The following operations all refer to an organization O ∈ ORGANIZATION described by relations is_org_described_by (O, Γ, Δ), has_basic_components (Γ, R, G, IL, ILL, ONT, M, ENV). This organization is modified by an operator, leading to a second organization O′ ∈ ORGANIZATION described by relations is_org_described_by (O′, Γ', Δ'), has_basic_components (Γ', R′, G′, IL′, ILL′, ONT′, M′, ENV′).

Our choice of primitive operators is motivated by different design guidelines and theories from social sciences [18,5,25], other disciplines, and our own research on formal modeling of organizations [9]. However, the application of the proposed set of operators is not restricted only to these theories. Thus, a designer has freedom to choose any sequence of operators for creating models of organizations. The operators are divided into three classes, which are consecutively described in the following subsections. Thus, in Section 3.1 the operators for roles are specified; in Section 3.2 the operators for different types of links are described; and in Section 3.3 the operators for groups are introduced.

3.1. Operators for roles

The classes of primitive operators for creating and modifying roles in a design object description for an organization are shown in Table 1.

A *role introduction operator* adds a new role to the organization. Usually, in organizational design after organizational tasks have been identified, these tasks should be further combined into positions (roles), based on the principles of labor division [23]. For example, in the conference organization setting, if the number of reviewers turns out to be insufficient, a Reviewer Recruiter role can be added to the Paper Selection role (see Fig. 2). This role, for example, may contact researchers to ask them to review for the conference by means of interaction with the environment.

3.1.1. Role introduction operator

Let op (O, O′, δ) be an operator that changes O into O′ with a focus on δ. Then op is a role introduction operator iff it satisfies:

1. $\delta \notin$ R, $\delta \in$ R′ such that is_role_in (δ, Γ')
2. structure_in_focus (O, ∅, ∅, ∅, ∅, ∅, ∅, ∅)
3. structure_in_focus (O′,{δ}, ∅, ∅, ∅, ONTf′, ∅, ∅), where ONTf′ = is_ontology_for (δ, o) and o ∈ ONT′

A *role retraction operator* removes all links connected to a role with their dynamic properties and mappings; it also deletes dynamic properties associated with the role and the role itself. In the example of the conference organization, when the Reviewer Recruiter has found enough reviewers, then the role can safely be removed from the organization.

3.1.2. Role retraction operator

Let op (O, O′, δ) be an operator that changes O into O′ with a focus on δ. Then op is a role retraction operator iff it satisfies:

1. $\delta \in$ R such that is_role_in (δ, Γ)
2. $\delta \notin$ R′
3. structure_in_focus (O, {δ}, ∅, ILf, ILLf, ONTf, Mf) ILf={e∈ IL|∃r′ ∈ R connects_to (e, δ, r′, Γ) ∨ ∃ r″ ∈ R connects_to (e, r″, δ, Γ)}
   ILLf={ill ∈ ILL|∃r ∈ R interlevel_connection (ill, δ, r, Γ)∨ ∃r′ ∈ R interlevel_connection (ill, r′, δ, Γ)}
   ONTf = is_ontology_for (δ, o), o∈ ONTMf={m ∈ M|∃ill ∈ ILLf has_onto_mapping (ill, m) ∨ ∃e ∈ ILf has_onto_mapping (e, m)}
4. structure_in_focus (O′, ∅, ∅, ∅, ∅, ∅, ∅)
5. dynamics_in_focus (O, DPf) DPf= {dp ∈ DP|has_dynamic_property (δ, dp) ∨∃e ∈ ILfhas_dynamic_property (e, dp)}
6. dynamics_in_focus (O′, ∅)

A *role dynamic property addition operator* creates a new property for the existing role in the organization and a *role dynamic property revocation operator* deletes a property from the dynamic description of a role.

3.1.3. Role dynamic property addition operator

Let op (O, O′, δ) be an operator that changes O into O′ with a focus on δ. Then op is a role dynamic property addition operator iff it satisfies:

Table 1
Operator classes for creating and modifying roles.

Class	Description
Role Introduction	Introduces a new role
Role Retraction	Deletes all links connected to a role with their dynamic properties and mappings; deletes a role and all dynamic properties associated with this role
Role Dynamic Property Addition	Adds a new dynamic property to a role
Role Dynamic Property Revocation	Deletes an existing role dynamic property

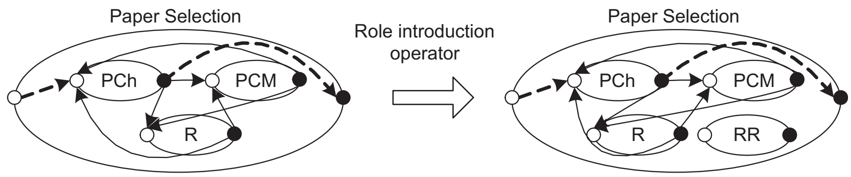


Fig. 2. Application of the role introduction operator for adding the Reviewer Recruiter role (RR) into the Paper Selection role.

1. dynamics_in_focus (O, $\emptyset$)
2. dynamics_in_focus (O′, DPf′) DPf′= $\{\delta \in DP' | \exists r \in R'$ has_dynamic_property (r, δ)}

3.1.4. Role dynamic property revocation operator

Let op (O, O′, δ) be an operator that changes O into O′ with a focus on δ. Then op is a role dynamic property revocation operator iff it satisfies:

1. dynamics_in_focus (O, DPf) DPf= $\{\delta \in DP | \exists r \in R$ has_dynamic_property (r, δ)}
2. dynamics_in_focus (O′, $\emptyset$)

3.2. Operators for links

In this subsection, we propose a set of classes of primitive operators for creating and modifying links in a design object description for an organization (see Table 2).

An *interaction link addition operator* allows the creation of an interaction link (information channel) between two existing roles in the organization. In the organizational design, after organizational subtasks are assigned to roles, the problem of coordination of interdependencies among subtasks should be solved.

In the conference management example, the Program Chair (playing in this case a managerial role) may request two reviewers to discuss their reviews. This requirement can be handled by the addition of interaction links between the appropriate reviewer roles in the design object description for an organization (see Fig. 3).

3.2.1. Interaction link addition operator

Let op (O, O′, δ) be an operator that changes O into O′ with a focus on δ. Then op is an interaction link addition operator iff it satisfies:

1. $\delta \notin$ IL, $\delta \in$ IL′ such that is_interaction_link_in (δ, Γ')
2. structure_in_focus (O, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$)
3. structure_in_focus (O′, $\emptyset$, $\emptyset$, $\{\delta\}$, $\emptyset$, $\emptyset$, Mf′, $\emptyset$) Mf′ = {m $\in$ M′|has_onto_mapping (δ, m)}

An *interaction link deletion operator* is used to delete an existing interaction link between two roles as well as to revoke all dynamic properties, associated with this link. For example, the Program Chair has taken care of the acceptance proceedings for the conference. He does not need to be in contact with the reviewers any more. This case can be handled by the deletion of the interaction between two roles in the design object description for an organization.

3.2.2. Interaction link deletion operator

Let op (O, O′, δ) be an operator that changes O into O′ with a focus on δ. Then op is an interaction link deletion operator iff it satisfies:

1. $\delta \notin$ IL′, $\delta \in$ IL such that is_interaction_link_in (δ, Γ)
2. structure_in_focus (O, $\emptyset$, $\emptyset$,$\{\delta\}$, $\emptyset$, $\emptyset$, Mf)
   Mf= {m $\in$ M| has_onto_mapping (δ, m)}
3. structure_in_focus (O′, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$)
4. dynamics_in_focus (O, DPf)
   DPf={dp $\in$ DP| has_dynamic_property (δ, dp)}
5. dynamics_in_focus (O′, $\emptyset$)

An *interaction dynamic property addition operator* creates a new property for an existing interaction link. An *interaction dynamic property revocation operator* deletes a property from the dynamic description of an interaction link.

3.2.3. Interaction dynamic property addition operator

Let op (O, O′, δ) be an operator that changes O into O′ with a focus on δ. Then op is an interaction dynamic property addition operator iff it satisfies:

Table 2
Operator classes for creating and modifying links.

Class	Description
Interaction Link Addition	Adds a new interaction link between any two roles
Interaction Link Deletion	Deletes an interaction link and all dynamic properties associated with this link
Interlevel Link Introduction	Introduces a new interlevel link
Interlevel Link Retraction	Retracts an existing interlevel link
Interaction Dynamic Property Addition	Adds a new dynamic property to an interaction link
Interaction Dynamic Property Revocation	Deletes an existing dynamic property, associated with an interaction link

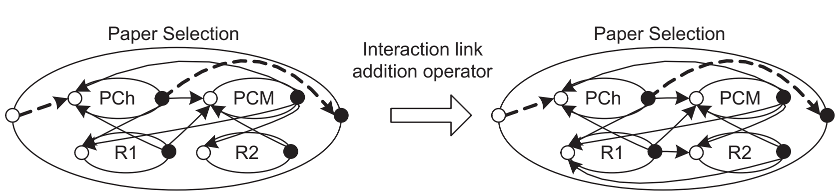


Fig. 3. Application of the interaction link addition operator for adding interaction links between Reviewer 1 role (R1) and Reviewer 2 role (R2) in the Paper Selection role.

1. dynamics_in_focus (O, $\varnothing$)
2. dynamics_in_focus (O′, DPf ′)
   DPf′= {$\delta \in$ DP′|∃e ∈ IL′ has_dynamic_property (e, δ)}

3.2.4. Interaction dynamic property revocation operator

Let op (O, O′, δ) be an operator that changes O into O′ with a focus on δ. Then op is an interaction dynamic property revocation operator iff it satisfies:

1. dynamics_in_focus (O, DPf)
   DPf = {$\delta \in$ DP|∃e ∈ IL has_dynamic_ property (e, δ)}
2. dymanics_in_focus (O′, $\varnothing$)

An interlevel link introduction operator creates a relation between a composite role and one of its subroles. It allows information that is generated outside the role to be passed into the role through its input interface or it allows information generated within a role to be transmitted outside through the role output interface. Normally, in hierarchical organizations decisions made at a managerial level are transferred to an operational level, e.g., to a certain department. Within the department this information is obtained by a certain role (s). For identifying which roles obtain this information, interlevel links are used. In the conference management example, the Conference Chair may have the possibility to send inquiries to Program Committee Members. This can be achieved by introducing an interlevel link between composite role Paper Selection (with which role Conference Chair has a direct connection by an interaction link) and its subrole Program Committee Member (see Fig. 4).

3.2.5. Interlevel link introduction operator

Let op (O, O′, δ) be an operator that changes O into O′ with a focus on δ. Then op is an interlevel link introduction operator iff it satisfies:

1. $\delta \notin$ IL, $\delta \in$ IL′ such that is_interlevel_link_in (δ, Γ)
2. structure_in_focus (O, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$)
3. structure_in_focus (O′, $\varnothing$, $\varnothing$, $\varnothing$, {δ}, $\varnothing$, Mf′, $\varnothing$) Mf′ = {m ∈ M′| has_onto_mapping (δ, m)}

An *interlevel link retraction operator* is used for breaking off interaction between some composite role and one of its subroles. This operation removes an interlevel link from the design object description for an organization. If the Conference Chair does not need to communicate with Program Committee Members any more, the interlevel link between these two roles can be retracted.

3.2.6. Interlevel link retraction operator

Let op (O, O′, δ) be an operator that changes O into O′ with a focus on δ. Then op is an interlevel link retraction operator iff it satisfies:

1. $\delta \notin$ IL′, $\delta \in$ IL such that is_interlevel_link_in (δ, Γ)
2. structure_in_focus (O, $\varnothing$, $\varnothing$, $\varnothing$, {δ}, $\varnothing$, Mf) Mf = {m ∈ M| has_onto_mapping (δ, m)}
3. structure_in_focus (O′, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$)

Table 3
Operator classes for creating and modifying groups.

Class	Description
Grouping	Combines roles into groups
Degrouping	Moves roles outside of a group and deletes the group
Group-to-Role	Transforms groups into roles
Role-to-Group	Transforms roles into groups

3.3. Operators for groups

The classes of primitive operators for creating and modifying groups in a design object description for an organization are shown in Table 3.

Often an organization designer can easily list a number of roles needed in an organization. However, it is not always clear which roles are related to each other, which roles would most often interact with each other, and so on. Once identified, the organization designer can group roles into sets.

In the literature on organizational design [28] different principles of grouping are described. For example, role grouping can be performed based on (1) similarities in role functional descriptions; (2) role participation in the same technological process; (3) identity or similarity of role technical specialties; (4) role orientation on the same market or customer groups. Often roles belonging to the same group interact with each other intensively. However, in the proposed organizational model, in contrast to roles, groups do not have interfaces. It means that every role within a group is allowed to interact with roles outside the group by means of direct interaction links. For example, in the conference organization the Program Chair and the Program Committee Members can be joined in one Program Committee group that will be responsible for making final decisions concerning paper acceptance. This can be accomplished by applying the grouping operator (see Fig. 5).

3.3.1. Grouping operator

Let op (O, Rg, O′, Gn) be an operator that changes O into O′ wrt. Gn ∈ G′, Rg ⊆ R. Then op is a *grouping operator* that creates a new group Gn from the subset of roles Rg iff it satisfies:

Structural aspect:

1. ∀a ∈ Rg member_of_in (a, Gn, Γ').
2. structure_in_focus (O, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$)
3. structure_in_focus (O′, $\varnothing$, {Gn}, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$, $\varnothing$)

Dynamic aspect:

1. dynamics_in_focus (O, $\varnothing$)
2. dynamics_in_focus (O′, DPf′)
   DPf′={dp ∈ DP′|has_dynamic_property (Gn, dp)}.
3. Er={e ∈ IL|∃r1 ∈ Rg ∃r2 ∈ Rg connects_to (e, r1, r2, Γ)}
   DPr = {dp ∈ DP|∃r ∈ Rg has_dynamic_property (r, dp) ∨∃e ∈ Er has_dynamic_property (e, dp)}
   DPg={dp ∈ DP′|has_dynamic_property (Gn, dp)}
4. DPg ⊆ DCL (DPr), where DCL (DPr) is the deductive closure of DPr

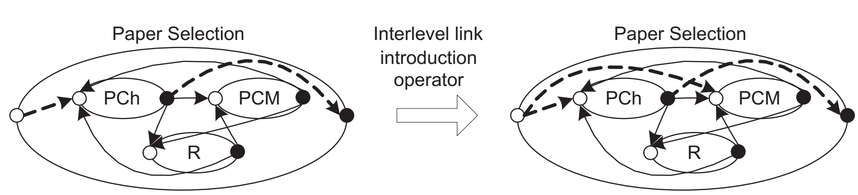


Fig. 4. Application of the interlevel link introduction operator for adding an interlevel link between Paper Selection role and Program Committee Member role (PCM).

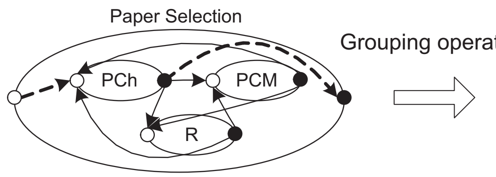


Fig. 5. Application of the grouping operator to create the Program Committee group that consists of roles Program Chair (PCh) and Program Committee Member (PCM) for making final decisions concerning paper acceptance.

A natural dual to the role grouping is role degrouping. This operator takes a group of roles and moves the roles to outside of the group. Role Degrouping transforms a group into a set of roles.

3.3.2. Degrouping operator

Let op (O, Gd, O′, Rdg) be an operator that changes O into O′ wrt. Gd ∈ G, and Rdg ⊆ R′. Then op is a *degrouping operator* iff it satisfies:

Structural aspect:

1. Rdg = {r ∈ R| member_of_in (r, Gd, Γ)}
2. Gd ∉ G′
3. structure_in_focus (O, ∅, {Gd}, ∅, ∅, ∅, ∅)
4. structure_in_focus (O′, ∅, ∅, ∅, ∅, ∅, ∅)

Dynamic aspect:

1. dynamics_in_focus (O, DPf) DPf = {dp ∈ DP|has_dynamic_property (Gd, dp) }.
2. dynamics_in_focus (O′, ∅)

A group can be transformed into a role, a more coherent, integrated and formal organizational unit with proper interfaces (e.g., a department of an organization). For a group to act as a role, it should have well-defined (formalized) input and output interfaces. A Group-to-Role operator takes a group and adds these interfaces. In an organic organization with loosely defined frequently changing structure this would correspond to the formalization of one of the organizational units, i.e., providing a formal (permanent) structural description with the subsequent specification of formal functional procedures. For example, in the conference organization setting, the Program Committee group from the Paper Selection role can be further transformed into the Program Committee role, a formal organizational unit with certain characteristics and functions (e.g., final decision making for the paper acceptance). Such transformation can be achieved by means of the Group-to-Role operator (see Fig. 6). The next logical step would be to limit the interactions of the subroles of the Program Committee role only to those that exist within the Program Committee role, and replace all interactions with the roles outside of the Program Committee role by corresponding interactions between outer roles and the Program Committee role. This can be done by applying interaction and interlevel link addition and retraction operators. In this case reviewers should follow a formal procedure for interactions with the Program Committee role and cannot directly address any arbitrary Program Committee Member.

3.3.3. Group-to-Role operator

Let op (O, g, O′, r) be an operator that transforms group g ∈ G in O into role r ∈ R′ in O′. Then op is a *Group-to-Role operator* iff it satisfies:

Structural aspect:

1. r ∉ R, g ∉ G′.
2. ∀a ∈ R member_of_in (a, g, Γ) ⇒ subrole_of_in (a, r, Γ').
3. structure_in_focus (O, ∅, {g}, ∅, ∅, ∅, ∅, ∅)
4. structure_in_focus (O′, {r}, ∅, ∅, ∅, ONTf′, ∅, ∅) ONTf′={o ∈ ONT′| has_internal_ontology (r, o) ∨ has_input_ontology (r, o) ∨ has_output_ontology (r, o)}

Dynamic aspect:

1. dynamics_in_focus (O, DPf) DPf={dp ∈ DP| has_dynamic_property (g, dp)}.
2. dynamics_in_focus (O′, DPf′) DPf′={dp ∈ DP′| has_dynamic_property (r, dp)}.
3. DPf ⇒ DPf′

A role may consist of several other roles that are not exposed to the rest of the world. When a role is converted to a group, it exposes the input and output interfaces of the roles inside it. Transforming a role into a group results in the subroles now residing on the level of the prior composite role. For example, during the reorganization some formal organization units (e.g., sections and departments) have been eliminated, whereas the roles that constituted these units and relations between them were kept, thus, creating a basis for new organizational formations.

3.3.4. Role-to-Group operator

Let op (O, r, O′, Gr) be an operator that changes O into O′, wrt. r ∈ R, and Gr ∈ G′. Then op is a *Role-to-Group operator* that transforms role r into group Gr iff it satisfies:

Structural aspect:

1. Gr ∉ G, r ∉ R′.
2. ∀a ∈ R subrole_of_in (a, r, Γ) ⇒ member_of_in (a, Gr, Γ').
3. structure_in_focus (O, {r}, ∅, ∅, ∅, ONTf, ∅) ONTf = {o ∈ ONT| has_internal_ontology (r, o) OR has_input_ontology (r, o) OR has_output_ontology (r, o)}
4. structure_in_focus (O′, ∅,{Gr}, ∅, ∅, ∅, ∅)

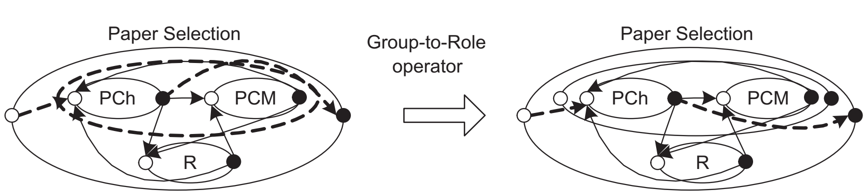


Fig. 6. Application of the Group-to-Role operator to transform Program Committee group into the Program Committee role.

Dynamic aspect:

1. dynamics_in_focus (O, DPf) DPf={dp ∈ DP| has_dynamic_property (r, dp)}.
2. dynamics_in_focus (O′, DPf′) DPf′={dp ∈ DP′|has_dynamic_property (g, dp)}.

4. Composing operators

The primitive operators described above reflect major principles of organizational design. In practice, in addition to the primitive operators more complex operators are used. Complex operators are represented as a combination of a certain number of primitive operators; some of them are given in Table 4.

Sometimes an effect produced by application of some composite operator to a design object description for an organization can be achieved by different combinations of primitive operators.

Consider the Role refinement operator as an example. This operator divides a role into several roles such that the role properties of the first role are distributed over the newer roles. In organizational design, role refinement corresponds to the fine-tuned specialization and division of labor for increasing efficiency. It is usually recommended to divide the work so that the portions be differentiated rather than similar, and that each role is responsible for a small portion of the overall task. According to Adam Smith, division of labor is limited by the extent of the market; other general principles of labor division can be found in [23].

Let us illustrate the application of the Role refinement operator in the context of the conference organizing example. In Fig. 7 the design object description for an organization is represented at the first aggregation level. Consider the situation when the decision is made to divide the tasks of Organizing Committee (OC) between the Local Organizing Committee (LOC), which is hence responsible for negotiations with publishers for printing proceedings and arranging the conference venue, and the General Organizing Committee (GOC), which is designated for solving financial and other questions. Thus, the role OC is refined into two new roles LOC and GOC. These roles are able to interact with each other and with the role Conference Chair.

Alternatively, every composite operator can be considered as an aggregated one-step operator. Such descriptions define formal conditions for a design object description for an organization before and after the application of a complex operator; therefore, they can serve the purpose of checking integrity and consistency of a design object description.

An example of such a representation for the Role refinement operator is given below.

4.1. Refinement operator (integrity definition)

Let op (O, r, O′, Rref) be an operator that refines role r ∈ R in O into a set of roles Rref ⊆ R′ in O′. Then op is a refinement operator iff it satisfies:

Structural aspect:

1. r ∈ R, r ∉ R′, Rref ∩ R = ∅
2. structure_in_focus (O, {r}, ∅, ILf, ILLf, ONTf, Mfa, ∅) ILf={e ∈ IL|∃r′ ∈ R connects_to (e, r′, r, Γ) OR ∃r″ ∈ R connects_to (e, r, r″, Γ)},
   ILLf = ILLfi ∪ ILLfo
   ILLfi={ill ∈ ILL |∃r′ ∈ R interlevel_connection (ill,r′,r, Γ)}
   ILLfo={ill ∈ ILL |∃r′ ∈ R interlevel_connection (ill,r,r′, Γ)}
   Mfa = Mf∪ Mfi∪ Mfo
   Mf= {m ∈ M|∃e ∈ ILf has_onto_mapping (e, m)}
   Mfi={m ∈ M|∃e ∈ ILLfi has_onto_mapping (e, m)}
   Mfo={m ∈ M|∃e ∈ ILLfo has_onto_mapping (e, m)}
   ONTf={o ∈ ONT| has_ontology (r, o)}

Table 4
Sample complex operators for creating and manipulating organizations.

Name	Pattern for	Description
Interaction Level Ascent	Interaction link deletion*. Role dynamic property addition*. Interlevel link addition*. Interaction link addition*.	Represents interaction between roles at a higher aggregation level
Interaction Level Descent	Interlevel link deletion*. Interaction link deletion*. Role interaction dynamic property addition*. Interaction link addition*.	A natural dual to Interaction Level Ascent operator
Role refinement	Role Retraction. Interlevel link deletion*. Interaction link deletion*. Interaction dynamic property addition*. Interlevel link addition*. Interaction link introduction*. Role dynamic property addition*. Role introduction*	Divides a role into several roles such that the role properties of the first role are distributed over the newer roles
Role join	Role Retraction*. Interlevel link deletion*. Interaction link deletion*. Interaction dynamic property addition*. Interlevel link addition*. Interaction link introduction*. Role dynamic property addition*. Role introduction	Joins several roles into a single new role
Adding aggregation levels	Interaction Level Ascent. Group-to-Role. Grouping. Role refinement*	Aggregates existing roles of the organization in more complex roles
Deleting aggregation levels	Degrouping.R-t-G. Interaction Level Descent	Replaces a composite role by a corresponding set of its constituent roles and relations between them
Regrouping	Grouping.Degrouping	Regroups the roles in an organization

The symbol * denotes that an operator can be applied zero, one or multiple times.

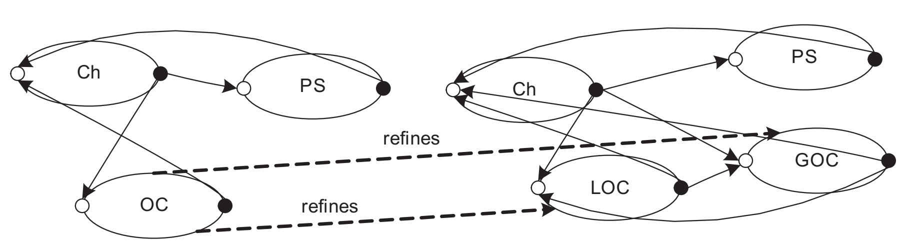


Fig. 7. Example of the application of the Role refinement operator, in which the Organizing Committee role (OC) is refined into the Local Organizing Committee (LOC) and General Organizing Committee roles (GOC).

3. $\text{structure\_in\_focus}\ (O', \text{Rref}, \varnothing, \text{ILf}', \text{ILLf}', \text{ONTf}', \text{Mfa}', \varnothing)$ $\text{ILf}' = \{e \in \text{IL}' \mid \exists r1 \in \text{Rref}\ \exists r2 \in \text{Rref}\ \text{connects\_to}\ (e, r1, r2, \Gamma')\ \text{OR}\ \exists r1' \in \text{Rref}\ \exists r2' \in R',\ r2' \notin \text{Rref}\ \text{connects\_to}\ (e, r1', r2', \Gamma')\ \text{OR}\ \exists r1'' \in \text{Rref} \exists r2'' \in R',\ r2'' \notin \text{Rref}\ \text{connects\_to}\ (e, r2'', r1'', \Gamma')\}$.
   $\text{ILLf}' = \text{ILLfi}' \cup \text{ILLfo}'$
   $\text{ILLfi}' = \{ill \in \text{ILL}' \mid \exists\ r' \in R'\ \text{interlevel\_connection}\ (ill, r', r, \Gamma')\}$
   $\text{ILLfo}' = \{ill \in \text{ILL}' \mid \exists r' \in R'\ \text{interlevel\_connection}\ (ill, r, r', \Gamma')\}$
   $\text{Mfa}' = \text{Mf}' \cup \text{Mfi}' \cup \text{Mfo}'$
   $\text{Mf}' = \{m \in M' \mid \exists e \in \text{ILf}'\ \text{has\_onto\_mapping}\ (e, m)\}$
   $\text{Mfi}' = \{m \in M' \mid \exists e \in \text{ILLfi}'\ \text{has\_onto\_mapping}\ (e, m)\}$
   $\text{Mfo}' = \{m \in M' \mid \exists e \in \text{ILLfo}'\ \text{has\_onto\_mapping}\ (e, m)\}$
   $\text{ONTf}' = \{o \in \text{ONT}' \mid \exists r1 \in \text{Rref}\ \text{has\_ontology}\ (r1, o)\}$.
4. $\forall e \in \text{IL}, \forall b \in R, b \in R', b \notin \text{Rref}\ \text{connects\_to}\ (e, r, b, \Gamma) \Rightarrow \exists e' \in \text{IL}', \exists r' \in \text{Rref}\ \text{connects\_to}\ (e', r', b, \Gamma')$ and
   $\forall e \in \text{IL}, \forall a \in R, a \in R', a \notin \text{Rref}\ \text{connects\_to}\ (e, a, r, \Gamma) \Rightarrow \exists e' \in \text{IL}', \exists\ r' \in \text{Rref}\ \text{connects\_to}\ (e', a, r', \Gamma')$.
5. $\forall e' \in \text{IL}',\ \forall\ r' \in \text{Rref}\ \forall b \in R'$ and $b \notin \text{Rref}\ \text{connects\_to}\ (e, r', b, \Gamma') \Rightarrow \exists e \in \text{IL},\ \text{connects\_to}\ (e, r, b, \Gamma)$ and
   $\forall e' \in \text{IL}',\ \forall\ r' \in \text{Rref}\ \forall a \in R'$ and $a \notin \text{Rref}\ \text{connects\_to}\ (e, a, r', \Gamma') \Rightarrow \exists e \in \text{IL},\ \text{connects\_to}\ (e, a, r, \Gamma)$.
6. $\forall a, a', m \in \text{Mfi}\ \text{is\_part\_of\_onto\_map}\ (a, a', m) \Rightarrow \exists a'', m' \in \text{Mfi}'\ \text{is\_part\_of\_onto\_map}\ (a, a'', m')$
   $\forall a, a', m' \in \text{Mfi}'\ \text{is\_part\_of\_onto\_map}\ (a, a', m') \Rightarrow \exists a'', m \in \text{Mfi}\ \text{is\_part\_of\_onto\_map}\ (a, a'', m)$
   $\forall a, a', m \in \text{Mfo}\ \text{is\_part\_of\_onto\_map}\ (a', a, m) \Rightarrow \exists a'', m' \in \text{Mfo}'\ \text{is\_part\_of\_onto\_map}\ (a'', a, m')$
   $\forall a, a', m' \in \text{Mfo}'\ \text{is\_part\_of\_onto\_map}\ (a', a, m') \Rightarrow \exists a'', m \in \text{Mfo}\ \text{is\_part\_of\_onto\_map}\ (a'', a, m)$

Dynamic aspect:

1. $\text{dynamics\_in\_focus}\ (O, \text{DPf})\ \text{DPf} = \{dp \in \text{DP} \mid \text{has\_dynamic\_property}\ (r, dp) \vee \exists e \in \text{ILf}\ \text{has\_dynamic\_property}\ (e, dp)\}$.
2. $\text{dynamics\_in\_focus}\ (O', \text{DPf}')\ \text{DPf}' = \{dp \in \text{DP}' \mid \exists r1 \in \text{Rref}\ \text{has\_dynamic\_property}\ (r1, dp)\ \text{OR}\ \exists e' \in \text{ILf}'\ \text{has\_dynamic\_property}\ (e', dp)\}$.
3. $\text{ONTp} = \{o \in \text{ONT} \mid \exists dp \in \text{DPf}\ \text{uses\_ont}\ (dp, o)\ \text{AND}\ o \notin \text{ONTf}\} \forall \varphi \in \text{DPEXP}$, such as $\text{uses\_ont}\ (\varphi, \bigcup_{o \in ONTp} o)\ [\text{DPf} \Rightarrow \varphi] \Rightarrow [\text{DPf}' \Rightarrow \varphi]$

A natural dual to the Role refinement operator is the *Role join operator*. This operator takes several roles and joins them into a single new role. Consider again the organization arranging a conference. If over time the differences between the tasks of the Program Committee Member and Reviewer roles disappear, then the roles Program Committee Member and Reviewer can be joined in one new role.

Let us consider one more often used complex operator *adding aggregation levels*. When certain roles have been joined in one group, this operator allows this group to be represented as an integral structural unit of an organization at the more abstract aggregation level. This operator has a counterpart in organizational design studies called *departmentalization*. Based on the departmentalization principles (cf. 19]) an organization is partitioned into structural units (called departments) with certain areas of responsibilities, a functional orientation, and a local authority power.

In the conference organization, the Adding aggregation levels operator can be applied for representing the Program Committee as an integral role that consists of the Program Chair and the Program Committee Member roles within the Paper Selection role. Such choice, for example, can be motivated by introducing a general formal procedure for paper acceptance. Hence, the Program Committee role is empowered (has a corresponding dynamic property) to make final decisions concerning paper selection. Applying the Aggregation Levels operator for this example can be considered as a three-step process (see Fig. 8 for the representation of the organization model (role Paper Selection) at aggregation level 2).

First, the roles Program Chair (PCh) and Program Committee Member (PCM) are joined into one group by applying the Grouping operator. After that, at step 2 the created group is transformed into the role Program Committee by adding interaction interfaces by means of the Group-to-Role operator. Finally, as the last step using the Interaction Level Ascent operator interaction links between roles PC and Reviewer (R) are created, as well as interlevel links within role PC.

5. Consistency of organizational specifications

To ensure internal consistency and validity of organizational specifications, specification constraints are identified, which can be checked automatically during the design process. The role of the constraints may differ in organization modeling which influences their format, purpose and way of use. Here we present a classification framework for constraints covering a range of

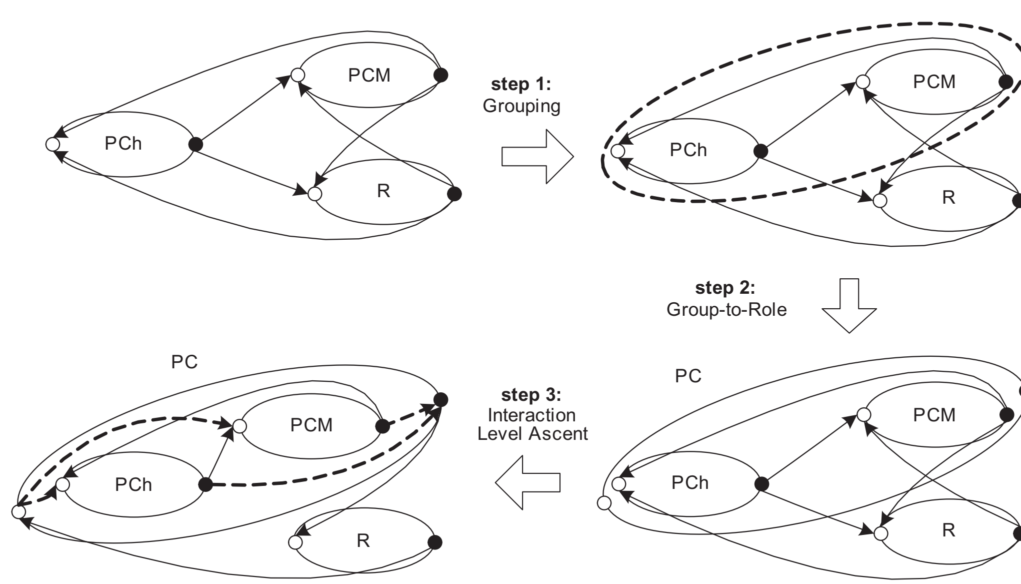


Fig. 8. Example of the application of the Adding Aggregation Levels operator, in which the roles Program Chair (PCh) and Program Committee Member (PCM) are grouped together and transformed into the Paper Selection (PC) role.

perspectives on organizations from very detailed to global, and from internal to external point of view, connecting the organization with its environment.

Specification constraints can be checked at every step of the design process in order to ensure the consistency and validity of the current specification. They can be classified based on their origin into: *generic constraints* that need to be satisfied by any organizational specification; *domain-specific constraints* dictated by the application domain of the specification.

Two types of generic constraints are considered: *structural consistency constraints* used to ensure consistency of the specification; *constraints imposed by the physical world* - the laws of the physical world render certain events, relationships between concepts, etc. impossible (e.g. a role cannot be at two locations at the same time).

The *consistency* of a specification is checked w.r.t. the set of structural consistency constraints. These constraints are axioms of the specification language and their logical consequences formulated based on the definitions of the language and reflecting the rules of correct and consistent use of the elements of the language in modeling. These constraints ensure internal integrity of the structures defined using the language. A specification S is consistent w.r.t. a set of structural consistency constraints SCC iff in each of its models each formula from SCC is true: $\langle I, v\rangle | = SCC$, where I is an interpretation of the sorts, functions and predicates of the language of S and v is a valuation of variables in S.

Consider two examples of structural consistency constraints for the organization-oriented view:

CS1: A role should receive only information types specified by its input ontology.

CS2: An information type outputted by some role should be related by the corresponding mapping to an information type from the ontology of the role-recipient.

Domain-specific constraints are imposed by the application domain in which the particular specification will be used and can be classified according to their sources:

Constraints imposed by the organization have been chosen (e.g. by the management of the company) as necessary and need to be satisfied by any specification for the particular organization. Such constraints can often be found in company policy documents, internal procedures descriptions, etc. *Constraints coming from external parties* are enforced by external parties (e.g. the society or government) and contain rules about working hours, safety procedures, emissions, etc. Sources for such constraints are regulations, agreements, etc. *Constraints of the physical world* come from the physical world w.r.t. the specific application domain and should be satisfied by any specification in this domain (in contrast to the generic physical constraints which should be satisfied by any specification irrespective of the application domain).

The *validity* of a specification is checked w.r.t. a set of physical-world and domain-specific constraints. An organizational specification S is *valid* w.r.t. a set of physical world and domain-specific constraints C iff in each of its models each formula from C is true: $\langle I, v\rangle | = C$.

To reduce the complexity of modeling and analysis, organizational specifications can be considered at different aggregation levels (e.g., to investigate certain organizational aspects, while abstracting from irrelevant details). To ensure consistency of specifications and sets of constraints of different aggregation levels, and integrity of a complete organizational specification, a set of *inter-level consistency constraints* is defined. A part of these constraints belong to the class of generic structural consistency constraints. For example,

CS3 : A role can be a subrole of one role at most.
In the structure $\Gamma\forall$r, r1, r2: ROLE subrole_of_in (r, r1, Γ) & subrole_of_in (r, r2, Γ)$\Rightarrow$ r2 = r1

CS4 : Each subrole of a composite role r should interact with at least one other subrole of r.
In the structure $\Gamma\forall$r1: ROLE subrole_of_in (r1, r, Γ) $\Rightarrow$ $\exists$r2:ROLE $\exists$e:INTERACTION_LINK subrole_of_in (r2, r, Γ) & (connects_to (e, r2, r1, Γ)| connects_to (e, r1, r2, Γ))

CS5 : Information provided to the input of a composite role should be further transmitted to one or more its subroles.

CS6: No role can be a subrole of itself at any aggregation level.

CS7: Information generated at the output of a composite role is transferred from the output of one of its subroles.

CS8: Any subrole of a composite role is not allowed to interact directly with any other role outside of this composite role.
The rest are domain-specific and should be identified and checked for a particular organization. For example:

CS9: Information of a type inf produced by a role r1 for a role r2 should be able to reach r2.

For checking if a path exists for communicating inf from r1 to r2 the following algorithm is proposed.

Algorithm 1. CHECK-EXISTENSE-OF-INTERACTION-PATH

```
1 i← max (AGR_LEVEL (r1), AGR_LEVEL (r2)), rt1 ← r1, rt2 ← r2
2 if ∃rh1, rh2 ∃Γ subrole_of_in (rt1, rh1, Γ) and
3     subrole_of_in (rt2, rh2, Γ) and rh1 = rh2,
4 then if IS_PATH_FROM_TO_FOR (rt1, rt2, inf) = true,
5           then return true, else return false.
6 if rt1 = r2 or rt2 = r1, then return true.
7 if AGR_LEVEL (r1) >= i and ∃rh1 ∃Γ subrole_of_in (rt1, rh1, Γ)
8 then if IS_PATH_FROM_TO_FOR (rt1, rh1, inf) = false
9      then return false.
10       rt1 ← rh1
11 if AGR_LEVEL (r2) >= i and ∃rh2 ∃ Γ subrole_of_in (rt2, rh2, Γ)
12 then if IS_PATH_FROM_TO_FOR (rh2, rt2, inf) = false
13        then return false.
14        rt2 ← rh2
15 i ← i – 1
16 until i > 0 perform steps 2–15.

Function AGR_LEVEL (r)

Output: returns the aggregation level number for role r
1 l ← 1, rt ← r
2 until ∃rh rh ≠ ORG and ∃Γ subrole_of_in (rt, rh, Γ), perform step
  3
3 rt ← rh, l ← l + 1
4 return l

Function IS_PATH_FROM_TO_FOR (src, dest, inf)
Output: returns true if a communication path exists from role src to
  role dest for information type inf, or returns false otherwise.
1 R ← {src}, RT ← ∅
2 R ← R ∪ RT
3 RT ← { r2 |∃e ∃r1 ∈ R ∃Γ connects_to (e, r1, r2, Γ)
4          and has_onto_mapping (e, inf, inf)}
5 if dest ∈ RT, then return true.
6 until RT ⊄ R, perform 2–5.
7 return false.
```

The general idea of the algorithm is to check if a communication path exists at every aggregation level, through which information is transferred on its way from the role-source to the role-destination. The algorithm begins from the maximum aggregation level

Table 5
Dynamics of the design process for the role refinement.

Actions of the designer	States of the tool
Chooses to address the role Organizing Committee (OC)	Proposes potentially applicable operators for role OC
Chooses the Role refinement operator	According to the specification of the Role refinement operator, initiates execution of role introduction operator and requests the designer to specify role names
Specifies GOC (General Organizing Committee) and LOC (Local Organizing Committee) names of the roles, into which role OC is refined	Requests to specify the elements of the ontologies for the newly created roles
Specifies the elements of the ontologies for roles LOC and GOC	Initiates execution of the role dynamic property addition operator. Requests to specify dynamic properties for the LOC and GOC roles
(optional) Specifies dynamic properties for the roles	Initiates execution of the interaction link introduction operator. Requests to specify interaction links between roles Chair (Ch), LOC and GOC
Specifies, which interaction links are needed between the roles	Initiates execution of the interaction dynamic property addition operator. Requests to specify dynamic properties for the introduced interaction links
(optional) Specifies dynamic properties for the interaction links	Initiates execution of the interaction link deletion operator, which removes all interaction links connected with role OC. Then, initiates execution of the role retraction operator, which removes role OC from the design object description

of both roles (Algorithm 1:1). Then the information flow is reconstructed gradually by proceeding from both ends (the source and the destination) simultaneously (from the source- Algorithm 1: 7–10, from the destination- Algorithm 1: 11–14) until the point is reached, at which the source-part flows directly into the destination part (Algorithm 1: 2–5 the parts connect at one level; 6: the parts connect across two levels). The worst case time complexity of the algorithm is estimated as $O(|\mathrm{LEVEL}| \cdot |\mathrm{LINK}|^2)$, where $|\mathrm{LEVEL}|$ is the number of aggregation levels, and $|\mathrm{LINK}|$ is the number of links between roles in the specification.

An instantiated version of this constraint with the template CS8(role1, role2, inf) for the conference organization case study considered in the paper is the following:

CS9(Conference Chair, Program Committee Member, inquiry): An inquiry about a paper sent by Conference Chair to Program Committee Member should be able to reach Program Committee Member.

6. A prototype tool to support the design of organizations

The formal representations of the organization's entities and the design operators described in this paper provide a solid basis for the development of a software environment supporting interactive organization design processes.

For the purpose of illustration and evaluation, a prototype tool was implemented, using the LEADSTO environment [7] and TTL Checking environment [8].[3] This tool supports organizational design and allows organization designers to investigate its dynamics. The LEADSTO tool was used to implement design operators and the design process. The TTL Checking tool was used to check the consistency of the organizational specification during the design process.

The application of the design prototype is demonstrated on the example of role refinement as described in the previous Section. The dynamics of the design process is described in Table 5. Parts of the corresponding design specification implemented in the tool are provided in Fig. 9. Based on this specification, a trace was generated automatically, a part of which is provided in Fig. 10 (the complete trace is provided in Appendix A).

A design specification consists of three essential types of elements: sorts, intervals and rules. Sorts (Fig. 9a) are used to define types of entities used in the design process (e.g., roles, design operators). Intervals are used to specify the input from the designer, external events and the initial design object description. For example, the interval in Fig. 9b specifies that the designer supports the choice of the Role refinement operator for role OC in ORG in the time interval [4, 5]. The design process and the design operators introduced in this paper are described using generic leadsto-rules. These rules are expressed in the form of direct temporal dependencies between two state properties in successive states. The format is defined as follows. Let α and β be state properties of the form 'conjunction of atoms or negations of atoms', and e, f, g, h non-negative real numbers. In the LEADSTO language the notation $\alpha \twoheadrightarrow_{e,\ f,\ g,\ h} \{\beta\}$ means: if state property α (antecedent indicated by A in Fig. 9c) holds for a certain time interval with duration g, then after some delay (between e and f) state property β (consequent indicated by C in Fig. 9c) will hold for a certain time interval of length h. The default time parameters are e = f = 0 and g = h = 1. For example, the rule in Fig. 9c generates possible operator alternatives, which could be applied to a role specified by the variable r.

In the design process, first, a designer chooses a part of the design object description, on which she intends to put her attention (in the considered example it is the role Organizing Committee). Next, the software proposes to the designer a number of operators, which are potentially applicable to the chosen part of the design object description. The designer chooses one of them, for the example, the Role refinement operator. Role refinement is a composite operator that consists of an ordered sequence of primitive operators. Usually, most of the primitive operators constituting composite ones are imperative (e.g., Role Introduction for Refinement); yet application of some of them may be postponed to the future (e.g., Role dynamic property addition for Refinement) or skipped (e.g., Interlevel link deletion for Refinement). Further, the tool demands specifying roles, into which role OC has to be refined. The designer specifies role names (for this example, Local Organizing Committee (LOC) and General Organizing Committee (GOC)) and their ontologies. At this step the software will check if the input ontology of role OC constitutes a subset of the union of the input ontologies elements of roles LOC and GOC. This is done by automated checking of the corresponding TTL property using the TTL Checking tool (see Fig. 11). Note that TTL properties can be defined in advance and be checked on the design specification at any step of the design process.

After that the software tool requests the designer to specify dynamic properties for the created roles. The designer may postpone this operation to a future point in time. Thereafter, the tool proposes to add interaction links between roles LOC, GOC and role Conference Chair (Ch), with which the original role OC was connected. At this step it is checked based on the integrity definition for refinement, whether the links, corresponding to the interaction links between Ch and OC in the original design object description, are present in the obtained design object description. Furthermore, if the original role had interlevel links with other roles, these links need to be deleted, and new interlevel links will be added. After

[3] Both environments can be accessed at http://www.few.vu.nl/~wai/TTL/; the LEADSTO specification for the design example considered in this section is available at http://www.few.vu.nl/~sharp/design.lt.

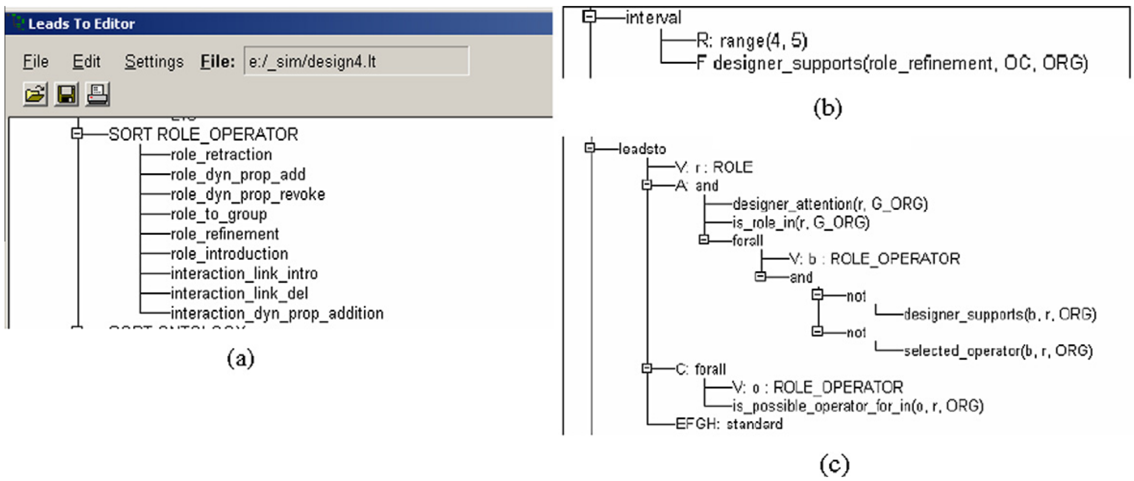


Fig. 9. Screen print of a design specification implemented in the tool comprising definitions of sorts (a), input from the designer in form of intervals (b), and generic rules describing the design process (in particular, design operators) (c).

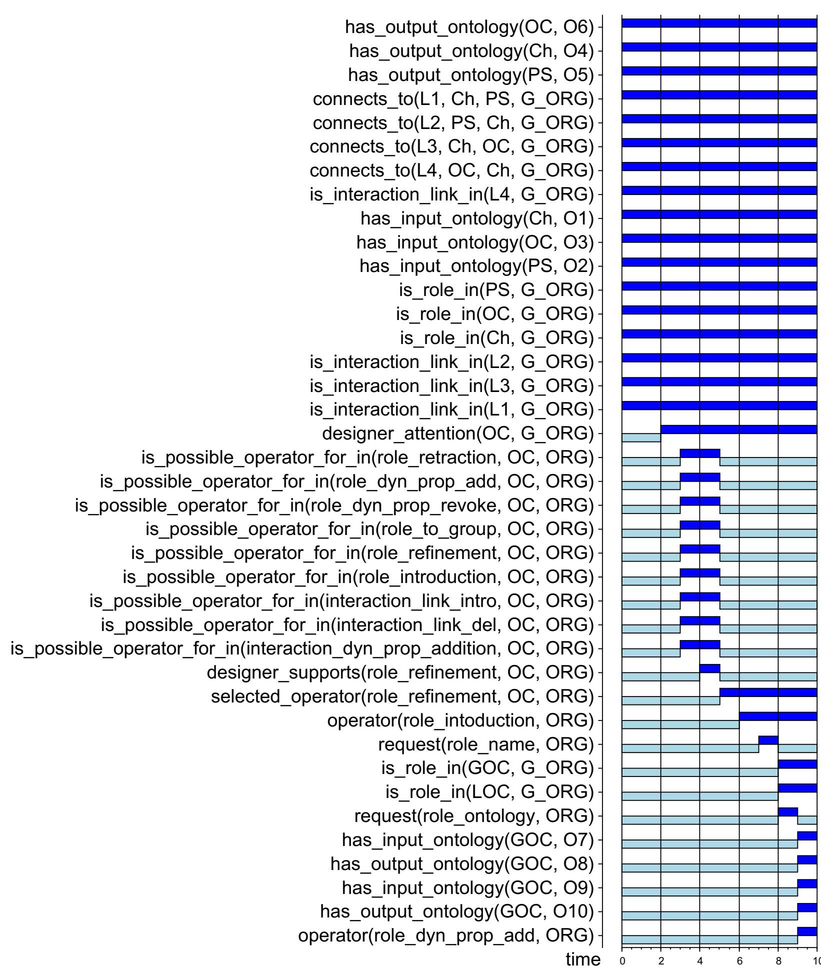


Fig. 10. Screen print of a partial trace illustrating dynamics of the design process for the role refinement; the darker line indicates that the state holds, the lighter line indicates that the state does not hold.

that the integrity constraints for the ontology mappings for these links need to be checked according to the integrity definition of the refinement operator. As the last step, role OC and interaction links connecting it with role Ch, as well as dynamic properties of role OC and its interaction links are automatically removed from the design object description.

7. Discussion

This paper introduces a representation format and a variety of operators for the design of organization specified in this representation format. The described operators have several important characteristics. First, they can be combined into composite oper-

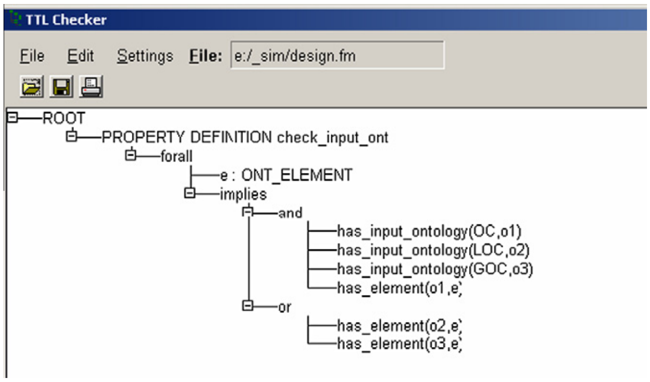


Fig. 11. The interface of the TTL Checking tool with the property specifying the consistency between the input ontology of role OC and the input ontologies of roles LOC and GOC, which replace OC.

ators that can serve as patterns for larger design steps in certain design cases. Second, the identified set of operators is independent of any organization theory or sociological methodology: they can be used for formalizing design principles from different theories. Third, a designer has freedom to choose any sequence of operators for creating designs of organizations of most types (e.g., functional and organic). An example of functional organizational design was discussed in this paper. When designing adaptive organic organizations, dedicated structural elements (e.g., the organization change management role) and dynamic descriptions (e.g., properties that describe the adaptation process) are specified. The operators offer both top-down refinements, as well as bottom-up grouping options. Finally, as has been shown the developed tool provides interactive support in designing organizations.

In comparison with other existing approaches which can be used for designing specifications of the organization-oriented view (e.g., UML structure and interaction diagrams [6], S-BPM ONE (2011)), the design approach proposed in this paper has the following advantages:

- it has a formal predicate logic-based basis (with properly defined syntax and semantics), which allows to define formal, dynamic, temporal properties of roles and links;
- the language TTL used to define properties allows: discrete and continuous temporal modelling of a system at different aggregation levels; numerical expressivity for modelling systems with explicitly defined quantitative relations best presented by difference or differential equations; specifying qualitative aspects of a system by expressing logical relationships between parts of a system;
- roles can be defined at different aggregation levels and properties of roles of different aggregation levels may be related to each other (see e.g., 23];
- basic operators proposed in the paper can be combined in composite operators based on patterns described in the literature on organizational design (some of such operators were described in Section 4);
- to ensure the consistency and validity of an organizational specification, automated checks can be performed at any step of the design process. For this the dedicated TTL Checking tool is used. To express properties to be checked on organizational specifications the whole expressiveness of TTL language can be used, including references to multiple time points, nested quantifiers and composite structures.

In this paper we focused on designing specifications of the organization-oriented view from the generic organization modeling and analysis framework [41]. In the future, the design of other views of the framework such as the process-oriented view [31] will be addressed as well.

To a certain extent organizations can be considered as compositional systems [48]. However, models and design methods for such systems do not allow representing many organization domain-specific concepts and operators (e.g., a group, a Group-to-Role operator) and, therefore, cannot capture many important organization phenomena.

In the area of component-based software engineering a number of design patterns for building software components (e.g., refinement, chaining, disjoint composition) have been introduced [20]. These patterns specify general-purpose manipulations with programming constructs (e.g., interface and private methods of components); while in organizational design literature organization transformations are described using domain-specific concepts. The formal representation format proposed in this paper bridges this gap and facilitates the abstraction of organization domain into general-purpose programming design patterns.

Formal specification of design processes enables verification of structural and dynamic consistency of a design object description for an organization. The verification of structural consistency is based on the consistency definitions for operators, such as one given in Section 4 for the Role refinement operator, and the consistency constraints described in Section 5. To check the consistency constraints, algorithms were developed and implemented, some of which were considered in the paper. For verifying dynamic consistency (e.g., checking relations between dynamic properties defined at different aggregation levels of a model representation) model checking techniques [27,43] may be used. Furthermore, verification mechanisms based on certain requirements on organizational functioning and performance (e.g., using organization performance indicators) were developed [33].

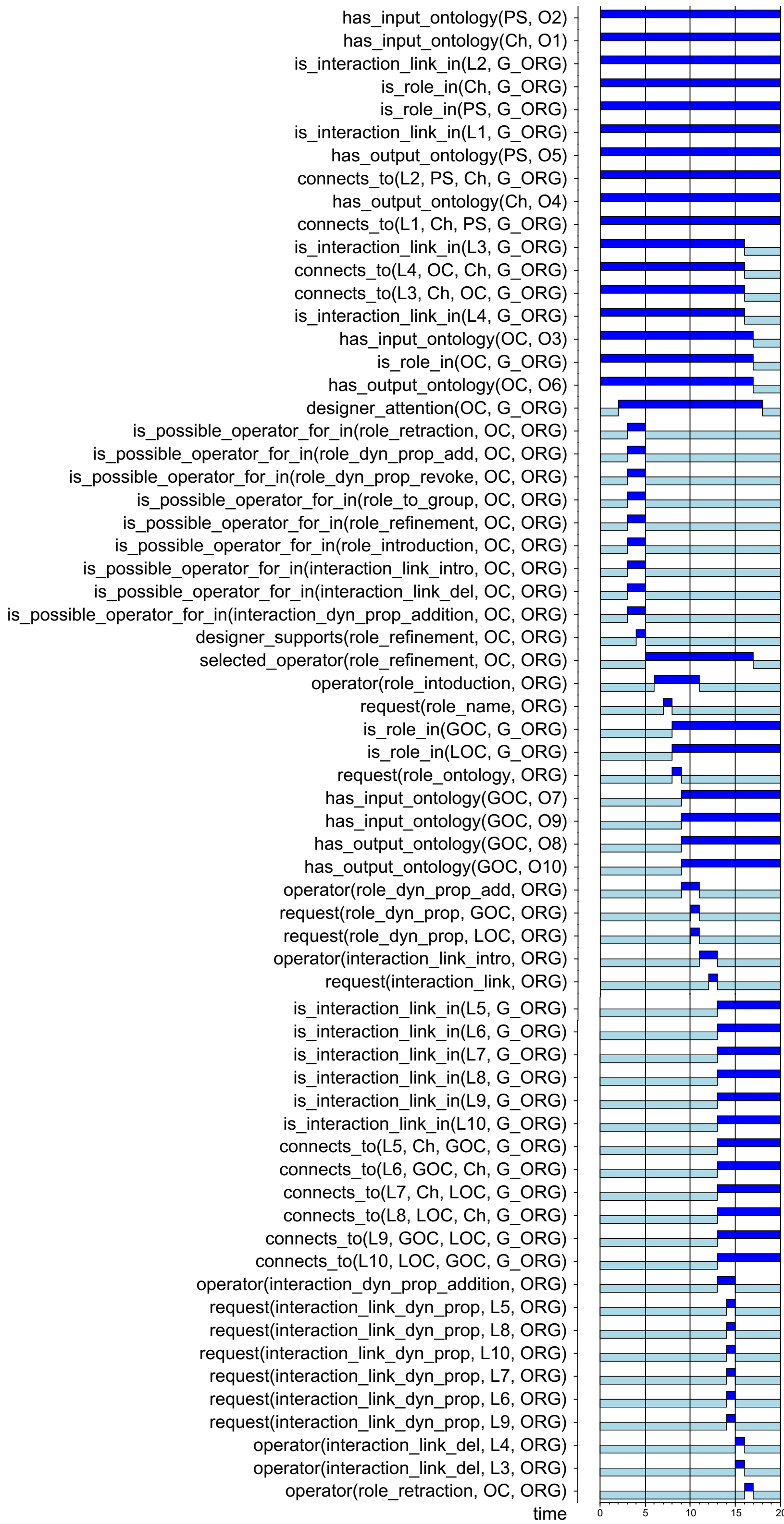
Another way to evaluate an organizational model is by performing simulations. For this purpose, agents with different types of attitudes and internal architectures may be allocated to roles within an organization model on certain conditions. After that, by considering different types and sequences of environmental influences provided within certain simulation scenarios, traces (i.e., temporal sequences of events in the environment and within the organization) corresponding to the execution of scenarios can be generated. These traces may be further used for analysis of the organizational model, more specifically, for evaluating different global properties of the organizational model (e.g., robustness, stability, efficiency, and effectiveness).

In conclusion, this paper introduced a representation format and a set of formally represented design operators dedicated to the design of organizations of most types. Although the choice of operators is motivated by different theories and guidelines from the area of organizational design, the application of the proposed operators is not restricted to any theories from social studies. The formalization of the operators provides a solid basis for the development of a software tool supporting interactive organization design processes. A prototype implementation for such a tool is demonstrated by an example in this paper.

Acknowledgments

This research is partly supported by the Dutch Technology Foundation STW, which is the applied science division of NWO, and the Technology Programme of the Ministry of Economic Affairs.

Appendix A. Screen print of a trace illustrating the dynamics of the design process for role refinement



References

[1] C. Argyris, D. Schön, Organizational Learning: A Theory of Action Perspective, Addison Wesley, Reading, Mass, 1978.
[2] C. Argyris, D. Schön, Organizational Learning II: Theory, Method and Practice, Addison Wesley, Reading, Mass, 1996.
[3] H. Barringer, M. Fisher, D. Gabbay, R. Owens, M. Reynolds, The Imperative Future: Principles of Executable Temporal Logic, Research Studies Press Ltd., John Wiley & Sons, 1996.
[4] E. Bertino, G.P. Zarri, B. Catania, Intelligent Database Systems, Addison-Wesley Professional, 2001.
[5] P.M. Blau, R.A. Schoenherr, The Structure of Organizations, Basic Books Inc., New York, London, 1971.
[6] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison Wesley, 1999.
[7] T. Bosse, C.M. Jonker, L. van der Meij, J. Treur, A language and environment for analysis of dynamics by simulation, The International Journal on Artificial Intelligence Tools 16 (2007) 435–464.
[8] T. Bosse, C.M. Jonker, L. van der Meij, A. Sharpanskykh, J. Treur, Specification and verification of dynamics in agent models, International Journal of Cooperative Information Systems 18 (1) (2009) 167–193.
[9] E. Broek, C. Jonker, A. Sharpanskykh, J. Treur, P. Yolum, Formal modeling and analysis of organizations, in: O. Boissier, V. Dignum, E. Matson, J. Sichman (Eds.), Proceedings of the Workshop on Organizations in Multi-Agent Systems, 2005, pp.17–33.
[10] J. Child, Organization: a choice for man, in: J. Child (Ed.), Man and Organization, Halsted Press, London, 1973, pp. 234–570.
[11] N. Chomsky, Aspects of the Theory of Syntax, The MIT Press, 1965.
[12] R.B. Duncan, What is the right organization structure?, Organizational Dynamics, Winter (1979) 59–79
[13] J. Dietz, Enterprise Ontology – Theory and Methodology, Springer-Verlag, Berlin, Heidelberg, 2006.
[14] ESPRIT Consortium AMICE (Eds.), CIMOSA: Open System Architecture for CIM. Berlin et al.: Springer, 1993.
[15] S. Feng, L.X. Li, L. Cen, An object-oriented intelligent design tool to aid the design of manufacturing systems, Knowledge-Based Systems 14 (5–6) (2001) 225–232.
[16] A. Fleischmann, C. Stary, Whom to Talk to? A Stakeholder Perspective on Business Process Management, Universal Access in the Information Society, Springer-Verlag, 2011. pp.1–26.
[17] U. Frank, Multi-perspective enterprise modeling (MEMO) – Conceptual framework and modeling languages, Hawaii international conference on system sciences, in: 35th Annual Hawaii International Conference on System Sciences (HICSS'02), vol. 3, 2002, pp. 1258–1267.
[18] J.R. Galbraith, Organization Design, Addison-Wesley Publishing Company, London, Amsterdam, Sydney, 1978.
[19] A. Habel, B. Hoffmann, Parallel independence in hierarchical graph transformation, in: Proceedings of International Conference on Graph Transformation, LNCS, vol. 3256, Springer-Verlag, Heidelberg, 2004, pp. 178–193.
[20] J. He, X. Li, Z. Liu, Component-based software engineering, in: D.V. Hung, M. Wirsing (Eds.), Theoretical Aspects of Computing, LNCS, vol. 3722, Springer, 2005, pp. 70–95.
[21] C.M. Jonker, A. Sharpanskykh, J. Treur, P. Yolum, A framework for formal modeling and analysis of organizations, Applied Intelligence 27 (1) (2007) 49–66.
[22] C.M. Jonker, J. Treur, A temporal-interactivist perspective on the dynamics of mental states, Cognitive Systems Research Journal 4 (3) (2003) 137–155.
[23] M. Kilbridge, L. Wester, An economic model for the division of labor, Management Science (1966) 255–269.
[24] K. Liu, L. Sun, J. Barjis, J.L.G. Dietz, Modelling dynamic behaviour of business organisations – extension of DEMO from a semiotic perspective, Knowledge-Based Systems 16 (2003) 101–111.
[25] J.W. Lorsch, P.R. Lawrence, Organization Design, Richard D. Irwin Inc., USA, 1970.
[26] M. Manzano, Extensions of First Order Logic, Cambridge University Press, 1996.
[27] K. McMillan, Symbolic Model Checking, Kluwer Academic Publishers, 1993.
[28] H. Mintzberg, Structure in Fives: Designing Effective Organizations, Prentice-Hall, NJ, 1993.
[29] I. Nonaka, R. Toyama, T. Hirata, Managing Flow: A Process Theory of the Knowledge-based Firm, Palgrave Macmillan, Basingstoke, New York, 2008.
[30] J. Pfeffer, Organizational Design, AHM Publishing Corp., Illinois, USA, 1978.
[31] V. Popova, A. Sharpanskykh, Process-oriented organisation modelling and analysis, Enterprise Information Systems Journal 2 (2) (2008) 157–176.
[32] V. Popova, A. Sharpanskykh, Modeling organizational performance indicators, Information Systems Journal 35 (4) (2010) 505–527.
[33] V. Popova, A. Sharpanskykh, Formal analysis of executions of organizational scenarios based on process-oriented specifications, Applied Intelligence Journal 34 (2) (2011) 226–244.
[34] V. Popova, A. Sharpanskykh, Formal modelling of organisational goals based on performance indicators, Data & Knowledge Engineering 70 (4) (2011) 335–364.
[35] R.F. Port, T. van Gelder (Eds.), Mind as Motion: Explorations in the Dynamics of Cognition, MIT Press, Cambridge, Mass, 1995.
[36] G. Rozenberg (Ed.), Handbook of Graph Grammars and Computing by Graph Transformation, Foundations, vol. 1, World Scientific, 1997.
[37] A.-W. Scheer, M. Nüttgens, ARIS architecture and reference models for business process management, in: W.M.P. van der Aalst, J. Desel, A. Oberweis, Business Process Management – Models, Techniques, and Empirical Studies, LNCS 1806, Berlin et al. 2000, pp. 366–379.
[38] L.C. Schmidt, J. Cagan, Recursive annealing: a computational model for machine design, Research in Engineering Design 7 (1995) 102–125.
[39] W.R. Scott, Organizations: Rational, Natural and Open Systems, Prentice Hall, USA, 1998.
[40] P.M. Senge, The Fifth Discipline: The Art and Practice of the Learning Organization, Doubleday, New York, 1990.
[41] A. Sharpanskykh, On Computer-Aided Methods for Modeling and Analysis of Organizations, PhD thesis, Vrije Universiteit Amsterdam, 2008a.
[42] A. Sharpanskykh, Authority and its implementation in enterprise information systems, Enterprise Information Systems Journal 4 (3) (2008) 66–79.
[43] A. Sharpanskykh, J. Treur, Verifying interlevel relations within multi-agent systems, in: Proceedings of the 17th European Conference on Artificial Intelligence, ECAI'06, IOS Press, 2006.
[44] A. Sharpanskykh, J. Treur, A temporal trace language for formal modelling and analysis of agent systems, in: M. Dastani, K. Hindriks, J.-J. Meyer (Eds.), Specification and Verification of Multi-Agent Systems (book chapter), Springer Verlag, 2010, pp. 317–352.
[45] G. Stiny, The algebras of design, Research in Engineering Design 2 (1991) 171–181.
[46] W.M.P. Van der Aalst, The application of Petri nets to workflow management, Journal of Circuits, Systems and Computers 8 (1) (1998) 21–66.
[47] W.M.P. Van der Aalst, Formalization and verification of event-driven process chains, Information and Software Technology 41 (10) (1999) 639–650.
[48] N. Wijngaards, Re-design of Compositional Systems, PhD Thesis, SIKS dissertation Series, 99-6, Vrije Universiteit Amsterdam, 1999.