A Formal Reuse-Based Approach for Interactively Designing Organizations

Catholijn Jonker, Jan Treur, and Pınar Yolum

Vrije Universiteit Amsterdam, Department of Artificial Intelligence
De Boelelaan 1081a, NL-1081 HV Amsterdam, The Netherlands
Email: {jonker, treur, pyolum}@few.vu.nl

Abstract. Multiagent organizations provide a powerful way for developing multiagent systems. This paper presents a methodology for designing organizations based on formal specification of requirements for organizational behavior and requirements refinement related to organizational structure. The approach allows parts of the organization to be designed in parallel and later be put together to satisfy the broader requirements of the organization. It is shown how designed organizational building blocks, can be formally specified, appropriately indexed and stored in an organization design library. The library structure, supported by software tools, allows designers with varying expertise to benefit by accommodating queries at different abstraction levels and by providing support for query reformulation.

1 Introduction

Organizations are an important metaphor for developing multiagent systems. Organizations provide a template of rules for agents to follow to accomplish large-scale tasks [Carley and Gasser, 1999]. When designed modularly, organizations make it possible to divide a large-scale task among small groups of practice and coherently put together the individual outputs of the groups to accomplish the large-scale tasks of interest. More specifically, by appropriately carrying out individual tasks and communicating as needed, organizations provide a way to solve large-scale tasks. We consider the problem of designing organizations. Such an organization design process for example starts by formally specifying requirements for the overall organization behavior. The requirements express the dynamic properties that should hold if appropriate organizational building blocks, such as groups and roles and their interactions, are glued together in an appropriate manner. In addition, there could be requirements on the structure of the desired organization that need to be fulfilled by the organization design. Given these requirements on overall organization (and, perhaps, some additional requirements), organizational structure and organizational behavior are designed and formally specified in such a manner that the requirements are fulfilled. However, designing the individual groups from scratch is labor-intensive, requiring expertise and domain knowledge.

We argue that once designed and formally specified, parts of an organization can be reused by other organizations. We propose a methodology for designing organizations based on reusing formal specifications of existing organizational components. The methodology indexes organizational components based on abstract identifiers that capture their functionality (what it does) and additional metadata that provide information on the workings of the component (how it does). An organization designer can interactively search a library of components to find a component that fits her needs and possibly tailor it to her needs. Since the components are indexed with identifiers taxonomically in different contextual dimensions, a designer can find the same component by formulating a query in a variety of ways. Further, the system is interactive in that it can exploit the library structure to help designers reformulate their queries more precisely.

The rest of this paper is organized as follows. Section 2 gives a technical background on the AGR methodology, which is used as a basis for the developed approach and the formal languages for specifying organizations and ontologies. Section 3 discusses our approach for designing organizations by refining requirements. Section 4 introduces the reuse methodology and discusses different indexing techniques for groups. Section 5 presents a methodology for classifying organizations based on multi-dimensional taxonomies. Section 6 discusses the relevant literature with respect to the proposed approach.

2 Technical Background

In this section some of the technical preliminaries are presented.

2.1 Agent/Group/Role Methodology

We start with the Agent/Group/Role (AGR) approach for modeling organizations [Ferber and Gutknecht, 1998]. This approach specifies a structure for an organization based on a definition of groups, roles and their relationships. An organization as a whole is composed of a number of *groups*. A group structure identifies the *roles* and the *transfers* between roles needed for interactions: the possible communication lines.

A group is a unit of communications. Two roles can communicate to each other if and only if they are in the same group. The *inter-group role interactions* (abbreviated as *group interactions*) between roles of different groups specify the connectivity of groups within an organization. *Agents* are allocated to roles; they realize the organization. However, the aim of an organization model is to abstract from specific agent allocations. Therefore instead of particular agents, roles are used as abstract entities, defining properties agents should have when they are to function in a given role within an organization.

We consider a negotiation group as a running example.

Example 1. Consider a buyer and a seller that need to agree on a price for an item. The seller proposes a price. The buyer can either accept or propose a different price. The seller can then accept or propose a new price. The process repeats itself until either the buyer or the seller accepts a proposed price.

The AGR approach to organization modeling has been extended to incorporate dynamic properties for the organization behavior in [Jonker and Treur, 2002].

C1 Role dynamic properties
Role dynamic properties relate input to output of that role. The input includes incoming communication from other roles as well as observations about the external world. The output includes outgoing communication as well as actions to be performed in the external world. The external world is considered as the environment of the organization that interacts with the organization by providing observational input to roles and which can be changed by actions in the output of roles. In general, the inputs and output of role properties capture public facts rather than private facts that are internal to an agent.

C2 Transfer dynamic properties
A transfer property relates output of the source role to input of the destination role. Typically, such properties express that information is indeed transferred from source to destination, and, for example, transfer is brought about within certain time duration. The parameters of the transfer property denote the roles that use this transfer. Intuitively, these roles should be uniquely identified.

C3 Group dynamic properties
Group dynamic properties relate input or output of roles within a group.

Example 2. A group property of the negotiation group explained before could be:

If at some point in time the Seller proposes a price,
then at some later time point the Buyer will receive an agreed, final price.

A special case of a group property is an *intragroup role interaction* property (RI) relating the outputs of two roles within a group. A role interaction property in this context always refers to roles in the same group.

C4 Group interaction dynamic properties
Group interaction properties relate the input of the source role in one group to the output of the destination role in another group. The same agent plays the two roles involved in a group interaction.

C5 Organization dynamic properties
Organization dynamic properties relate to input and/or output of roles within the organization. A typical (informal) example of such a property is: 'if within the organization, role A promises to deliver a product, then role B will deliver this product'.

Table 1 provides an overview of these combinations. Group interaction properties can be considered a specific type of organization property. Similarly, role interaction and transfer properties can be considered a specific type of group properties. Note that with respect to simulation, the above dynamics definition can contain elements that are redundant: a smaller subset of dynamical properties can form an executable specification of the dynamics of an AGR type organization. For example, on the basis of the roles, transfer properties, and group interactions, the organization can be simulated. The group dynamic properties, including the role interaction properties, and the organization properties should emerge in the execution, and testing for them can validate the model.

Property type	Notation	Relating	
Organization	OP	Input or Output of	roles in O
Group interaction	GI	Role r1 Input in G1	Role r2 Output in G2
Group	GP	Input or Output of	roles in G
Role interaction	RI	Role r1 Output in G	Role r2 Output in G
Transfer	TP	Role r1 Output in G	Role r2 Input in G
Role	RP	Role r Input [Role r Internal]	Role r Output

Table 1. Types of dynamic properties for an AGR organization model

2.2 A Formal Specification Language for Organization Structure

In this paper we use a subset of the formal language developed for specifying the structure and behavior for AGR-models of organizations; cf. [Jonker and Treur, 2002]. This language is used to specify the properties explained in Section 2.1.

Sort	Description
ROLE	Sort for a role within an organization.
AGENT	Sort for an agent that can be allocated to a certain role.
GROUP	Sort for a group within an organization.
GROUP_INTERACTION	Sort for a connection between two roles in different groups
TRANSFER	Sort for a connection between two roles within one group.
CONNECTION	An element of TRANSFER or GROUP_INTERACTION
DYNPROP	Sort for names of dynamic properties
DYNPROPEXP	Sort for possible TTL expressions (see Section 2.4)

Table 2. Sorts of the language

Table 2 gives an overview of the possible sorts to specify the elements of an organization. From a structural perspective, some of these sorts relate to the each other through the predicates of Table 3. These predicates specify the groups in the organizations, the roles in the groups, the agents allocated to these roles, and the communication between two roles.

Predicate	Description
exists_role: ROLE	A role exists within an organization.
exists_group: GROUP	A group exists within the organization
role_belongs_to_group: ROLE * GROUP	A role is part of a group.
intra_group_connection: ROLE * ROLE * GROUP * TRANSFER	A role is connected to another role (directed) within a certain group by means of a transfer connection. The source and destination roles are allowed to be equivalent.

Table 3. Predicates for specifying the structure of an organization

The predicates in Table 4 are used to define the behavior of the organization. These properties essentially specify the role, group, and organization properties as well as the interaction properties between groups. Modeling the behavior of an organization makes use of dynamic properties expressed in terms of the Temporal Trace Language TTL. The different types of properties are defined in Table 3.

has_expression: DYNPROP * DYNPROPEXP	A specific dynamic property has an expression.

Table 4. Predicates for specifying the behavior of an organization

2.3 A Formal Language for Ontologies

In addition to the language described above we developed a language to formally specify ontologies for the input and output states of roles. This language is used to describe the ontology. The language is based on first-order many-sorted logic; e.g., [Meinke and Tucker, 1993].

Definition 1. A signature Σ is a four tuple **<S, C, F, P>** such that

- **S** is a set of sorts
- **C** is a set of constants, which have sorts defined in **S**
- **F** is a set of functions with possibly varying arity and whose domain and range elements have sorts defined in **S**
- **R** is a set of relations with possibly varying arity and whose domain elements have sorts defined in **S**

Table 1 provides the constructs of the language. Here, we describe them briefly. The ontology can refer to many sorts. Let s1...sn denote sorts and o1...on denote ontologies. The sorts of the ontology are those for which the is_a_sort_in(s_1, o_1) predicate holds. Let r1...rn denote relations and $f_1...f_n$ denote functions. The relations in the ontology are shown with is_a_relation(n, r_1, o_1), where n denotes the arity of relation r_1. Similarly, the functions in the ontology are shown with is_a_relation(n, f_1, o_1), where n denotes the arity of function f_1. For each relation in the ontology, dom_of(n. s_1, r_1) specifies the domain sorts for each of the n parameters in the relation. Similarly, for each function in the ontology, dom_of(n, f_1, r_1) gives the domain sort for all n parameters for the function. For the functions, we also define the range_of(f_1, r_1), which give the range sort for the function.

PREDICATE	**DESCRIPTION**
is_a_sort_in:SORT * ONTOLOGY	SORT exists in ONTOLOGY
is_a_relation_in:INTEGER*RELATION*ONTOLOGY	RELATION exists in ONTOLOGY with arity n
is_a_function_in:INTEGER*FUNCTION* ONTOLOGY	FUNCTION exists in ONTOLOGY with arity n
Dom_of: INTEGER*SORT * RELATION	Domain of RELATION is in SORT
Dom_of: INTEGER*SORT * FUNCTION	Domain of FUNCTION is in SORT
Range_of: SORT * FUNCTION	Range of FUNCTION is in SORT

Table 5. Basic elements of a language for signatures

These predicates allow us to formally specify an ontology in the form of signatures. Given such a signature, one can also define well-formed formulae as follows.

Definition 2. Let Σ = <**S, C, F, P**> be a signature and **V** a set of variables with sorts defined in **S**. The set of well-formed formulae over Σ, WFF(Σ) are generated the as usual.

2.4 A Formal Specification Language for Organization Behavior

To formally specify dynamic properties characterising organization behavior, an expressive language is needed. In this paper for most of the occurring properties both informal or semi-formal and formal representations are given. The formal representations in the sort DYNPROPEXP (see Section 2.2) are based on the Temporal Trace Language(TTL; cf. [Jonker and Treur, 2002]), which is briefly defined as follows.

A *state ontology* is a specification (in order-sorted logic) of a vocabulary, i.e., a signature. A state for ontology Ont is an assignment of truth-values {true, false} to the set At(Ont) of ground atoms expressed in terms of Ont. The *set of all possible states* for state ontology Ont is denoted by STATES(Ont). The set of *state properties* STATPROP(Ont) for state ontology Ont is the set of all propositions over ground atoms from At(Ont). A fixed *time frame* T is assumed which is linearly ordered. A *trace* or *trajectory* $\mathcal{T}$ over a state ontology Ont and time frame T is a mapping $\mathcal{T}$: T → STATES(Ont), i.e., a sequence of states $\mathcal{T}_t$ (t ∈ T) in STATES(Ont). The set of all traces over state ontology Ont is denoted by TRACES(Ont). Depending on the application, the time frame T may be dense (e.g., the real numbers), or discrete (e.g., the set of integers or natural numbers or a finite initial segment of the natural numbers), or any other form, as long as it has a linear ordering. The set of *dynamic properties* DYNPROPEXP(Σ) is the set of temporal statements that can be formulated with respect to traces based on the state ontology Ont in the following manner (for an organization or part thereof, Ont is the union of all input, output and internal state ontologies of the roles in the organization (part).

Given a trace $\mathcal{T}$ over state ontology Ont, the input state of a role at time point t is denoted by state($\mathcal{T}$, t, input(r)); analogously, state($\mathcal{T}$, t, output(r)), and state($\mathcal{T}$, t, internal(r)) denote the output state and internal state of the role.

These states can be related to state properties via the formally defined satisfaction relation |=, comparable to the Holds-predicate in the Situation Calculus: state($\mathcal{T}$, t, output(r)) |= p denotes that state property p holds in trace $\mathcal{T}$ at time t in the output state of the organism. Based on these statements, dynamic properties can be formulated in a formal manner in a sorted first-order predicate logic with sorts T for time points, Trace for traces and F for state formulae, using quantifiers over time and the usual first-order logical connectives such as $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\forall$, $\exists$.

3 Designing Organizations by Requirements Refinement

Consider an organization design problem for which the requirements of the overall behavior are given in the form of dynamic properties. In other words, the organization designed for this problem should at least satisfy certain these given properties. One approach for designing such an organization is a top-down approach. The design process starts from these global organization properties. The properties are then refined into a set of smaller properties that can be satisfied by parts of the organization. Hence, the design problem is reduced to designing correct groups that can satisfy some properties and establishing effective communications between these groups. This refinement process is captured via the following logical scheme.

dynamic properties for the groups &
dynamic properties for group interaction $\Rightarrow$ dynamic properties for the organization

Example 3. Consider an organization where a Buyer chooses an item and then the Buyer and Seller agree on a price. This organization can be designed by first obtaining (designing from scratch or reusing) two groups as follows. The first group will communicate in a certain way that will allow the Buyer to choose an item. The second groups functions as described in Example 1. When these groups are linked correctly (by a group interaction) then the overall requirements of the organization are satisfied.

An important aspect of this approach is its formality. The informal group property of Example 2, where a Buyer and a Seller negotiates of the price of a commodity, can be formalized as follows. In this and the following examples, a communication ontology is used. The `communication_from_to` predicate is used to describe the roles that communicate, the type of the communicative act, and the content of the act. An alternative would be to to use the content of the communicative act by itself to denote the states. Further, the communication ontology could be replaced by other ontologies, for example with a service ontology.

Example 4. Let `communication_from_to(Role1, Role2, inform, desired_price_for(a))` denote that `Role1` is informing `Role2` that it is interested in agreeing on a price for item `a` and `communication_from_to(Role1, Role2, inform, agreed_price_for(p, a))` denote that `Role1` is informing `Role2` that it is agreeing to the price `p` for item `a`. Then the following TTL formulation of a group property means that if the Buyer and the Seller roles inform each other on a desire to agree on a price, then at a later time they will agree on a price.

$\forall$t [state(γ, t, output(Buyer)) |= communication_from_to(Buyer, Seller, inform, desired_price_for(a)) &
state(γ, t, output(Seller)) |= communication_from_to(Seller, Buyer, inform, desired_price_for(a))
$\Rightarrow$

∃t'≥t ∃p
state(γ, t', input(Buyer)) |= communication_from_to(Seller, Buyer, inform, agreed_price_for(p, a)) & state(γ, t', input(Seller)) |= communication_from_to(Buyer, Seller, inform, agreed_price_for(p, a))]

The refinement scheme shows that to fulfill the overall dynamic properties, dynamic properties of certain groups and group interactions together imply the organization behavior requirements. The process to determine the requirements for parts of the organization and groups is called *requirements refinement.* It provides new, refined requirements for the behavior of groups and group interaction. It is possible to arrive at requirements of groups in one step, but it is also possible to first refine requirements for the behavior of the organization as a whole to the requirements on the behavior of parts of the organization, before further refinement is made to obtain dynamic properties for groups. Notice that the groups are not given at forehand, but his requirements refinement process just determines which types of groups (i.e., with which properties) are chosen as part of the organization being designed.

Similarly, the required dynamic properties of groups can be refined to dynamic properties of certain roles and transfers, making use of the following scheme:

dynamic properties for roles &
dynamic properties for transfer between roles ⇒ dynamic properties for a group

This provides the roles to be used, requirements on the behavior of these roles and transfer between them, that together imply the requirements on the behavior of the group. Again it is possible to first refine requirements on the behavior of a group to requirements of the behavior of parts of the group, before further refinement to required role behavior is made.

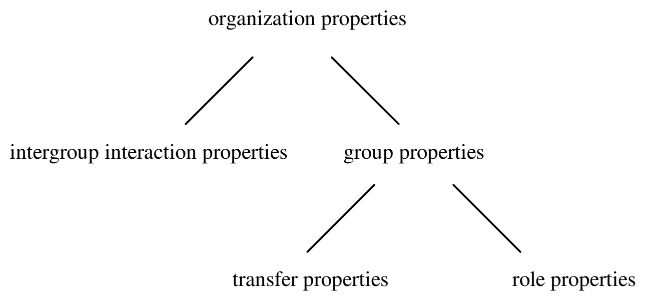

Figure 1. Inter-level relations between dynamic properties

An overview of the inter-level relationships between these dynamic properties at different aggregation levels is depicted as an AND-tree in Figure 1. In summary, from the design perspective, a top-down refinement approach can be followed. That is, the requirements on overall organizational behavior can be first refined to requirements on behavior of groups and group interaction, and then the requirements on behavior of groups can be refined to requirements on roles and transfers. This design perspective may suggest that designing organizations always has to be done from scratch. However, as often parts of organizations can be used in other organizations, an organization design process can benefit substantially if reusable parts of organization models are maintained in a library and indexed in an adequate manner to retrieve relevant ones during a given design process. The methodology for organization design based on an extension of the

AGR model that has been developed supports reuse of organization parts and, in particular, of groups. This will be addressed in subsequent sections.

4 Indexing Within the Library of Groups

The approach to reuse within the developed organization design methodology is inspired from the literature on reuse in software design [van Vliet, 2000]. The main steps of the approach that are related to the reuse of groups are the following:

1. Groups are characterized from an external perspective by abstract identifiers at different levels of abstraction.
2. The complete group specification (from an internal perspective) is stored in the library, and indexed with the identifiers obtained in 1.
3. An organization designer queries the library for a group based on certain information expressed in terms of the characterizing identifiers.
4. The library returns all groups that match the query, based on a matching function.
5. The organization designer reviews the returned groups and incorporates one of them possibly modifying it as necessary.

The manner in which (formal) specifications of the internal structure of groups are stored in the library is shown in Section 2. It is assumed that these groups are indexed by identifiers at different levels of abstractions according to multi-dimensional taxonomic structures, one taxonomy for each dimension that can be considered in the query. This section studies different methods of indexing groups. Section 5 shows how these indexes can be combined into a multi-dimensional taxonomy to allow flexible queries.

4.1 Indexing by Group Functionality, Output or Input

A first dimension to consider for indexing is indexing on the basis of a group's functionality. Such a functionality can be considered as a relation between properties of input states and of output states of a group. The simplest form is that a certain input state property at a certain point in time leads to a certain output state property at some later time point. However, especially for ongoing processes, the relationships between input and output state properties can take the form of more complex temporal relationships, expressed by dynamic properties as discussed in Section 2. For simple primary functionality descriptions, there may be other secondary functionality descriptions involved, such as other constraints on how the group should carry out this primary functionality. Often, there are restrictions on the group's process that have to be satisfied. One form of constraints requires that a certain condition is maintained throughout the group enactment. For example, in our running example, a designer could additionally require that the no role should announce a final price p for an item a, without getting the permission of the second role. This is a constraint that should be maintained at any time in the group. The temporal trace language TTL (see Section 2) can be used to specify such dynamic properties as part of a functionality description.

Example 5. Let the previously defined communications carry their meanings and let `communication_from_to(Role1, Role2, permit, announce_price)` mean that `Role1` gives a permission to `Role2` to announce the agreed price. The following TTL formulation then expresses the following property. If both Buyer and Seller inform the other for a desire on agreed price, then at a later time they will communicate each other the same agreed price. And if either the Buyer or the Seller announces the price to a third-party C, then that will be done with the other party's permission.

∀t [state(γ, t, output(Buyer)) |= communication_from_to(Buyer, Seller, inform, desired_price_for(a)) &
state(γ, t, output(Seller)) |= communication_from_to(Seller, Buyer, inform, desired_price_for(a))
⇒
∃t'≥t ∃p
state(γ, t', input(Buyer)) |= communication_from_to(Seller, Buyer, inform, agreed_price_for(p, a)) & state(γ, t', input(Seller)) |= communication_from_to(Buyer, Seller, inform, agreed_price_for(p, a))]
&
∀t" ∀C state(γ, t", input(Buyer)) |= communication_from_to(Buyer, C, inform, agreed_price_for(p, a)) ⇒
∃t'"≤ t"
state(γ, t'", input(Buyer)) |= communication_from_to(Seller, Buyer, permit, announce_price))
&
∀t" ∀C state(γ, t", input(Seller)) |= communication_from_to(Seller, C, inform, agreed_price_for(p, a)) ⇒
∃t'"≤ t"
state(γ, t'", input(Seller)) |= communication_from_to(Buyer, Seller, permit, announce_ price))

Recall that the group properties as defined above from an internal viewpoint relate input properties to the output properties of roles within a group. One (most specific) way to identify a group is to use such a dynamic group property of a group as an identifier. The advantage of this group property is that it captures the functionality of the group succinctly. The main disadvantage of its usage as an identifier, however, is that it contains too much internal information to sufficiently abstract the group. A designer in need of a group would have to know the roles in this group as well as the exact inputs and outputs of the roles to retrieve the group from the library. For example, if a designer had used the group property above to search for a group, then the designer would need to know that the group contains at least two roles of Seller and Buyer and their inputs and outputs. Since many designers would not know this much internal information about a group, searching a group through such group properties will not be too useful.

Next, we study two opposite ways that the groups can be identified. On one extreme, we have designers who know precisely how the group they are looking for should behave from an external viewpoint, but do not know, and do not care for, the internal details of the group. For these designers, a group identifier is created that captures the externally observable functionality of the group in detail, but abstracts from the internal details of the group. More specifically, we start with the group property (specified from an internal viewpoint) considered above to derive from it a more abstract property (specified from an external viewpoint). Following the same example, the instantiation of a group would have carried the organization from a state of where participants declared interest to reach an agreed price to a state where they have reached the agreed price. Intuitively, this functionality can be specified in an abstract group identifier that captures the information on interest and agreed price in respective input and the output states of the group, but does not refer to any specific roles or information transfer inside the group. As an example consider the output state of this group; i.e., there is an agreed price. This output state (OS) or the input state (IS) can be defined using certain signatures Σin and Σout, and state properties Win and Wout, respectively. These signatures can be taken as copies or abstracted forms of the internal signatures related to some of the roles (see also Example 10 below). For simplicity we assume that the internal signatures for the roles are disjoint, so that there is a one-to-one correspondence between signatures used at group input or output states and some of the internal signatures. Given the group input and output signatures, the abstract group identifier relates group input states and group output states, and abstracts from specific role names.

Example 6. Recall the group property of Example 4 where the Buyer and Seller eventually agree on a price. The property here uses that of Example 4 as a basis but the specification is now done from an external point of view. The specification does not refer to the Buyer and Seller roles but to variable roles X and Y.

∀t [state(γ, t, input(G)) |= ∃ X, Y:ROLE communication_from_to(X, Y, inform, desired_price_for(a)) & communication_from_to(Y, X, inform, desired_price_for(a))
⇒ ∃t'≥t ∃p
state(γ, t', output(G)) |= ∃ X, Y:ROLE communication_from_to(X, Y, inform, agreed_price_for(p, a)) & communication_from_to(Y, X, inform, agreed_price_for(p, a))]

A group input state carries information obtained from outside the group, whereas a group output state carries information targeted for the outside of the group. In Example 8, the information on the input to group G, expresses that some X initiates a communication of a certain type to some Y. Similarly, the group output state carries information that results from a communication of some X and Y. Note that the X and Y in this dynamic property may be related to other roles (other than Buyer and Seller) in other parts of the organization, for example Standkeeper and Visitor. When incorporating the negotiation group, by an intergroup role interaction these other roles will be connected to the Buyer and Seller role. Not all descriptions of group functionality have the simple form that one input state property `Win` after some time will lead to an output state property `Wout`. It may well be the case that a group has as functionality that it is adaptive, in the sense that, depending on the amount of work it does, its functioning is improving over time. For such a group a dynamic property of the format 'exercise improves skill' can be expressed in TTL:

Example 7. In the following TTL expression, state(γ1, t', input(G)) |= has_work_level(v1) means that in trace γ1 at time t', at the input of the group, the group has work level v1. The TTL expression then means that for every pair of traces γ1 and γ2, if over a certain time interval a trace has a higher work level than a second trace, then after this time interval the first trace will perform with a higher quality level.

∀γ1, γ2, t
[∀t'≤t [state(γ1, t', input(G)) |= has_work_level(v1) & state(γ2, t', input(G)) |= has_work_level(v2) ⇒ v1 ≤ v2] &
state(γ1, t', output(G)) |= has_quality_level(w1) & state(γ2, t', output(G)) |= has_quality_level(w2)
⇒ w1 ≤ w2]

Notice that this property, expressible in TTL, is more complex both in the temporal structure and in the fact that two possible traces are compared (which is not possible, for example, in standard temporal logics).

An abstract group identifier has the advantage of identifying a group with a group property (i.e., captures the functionality well) but also has the advantage of only capturing the externalized functionality of the group without referencing any internal roles or information flow.

On the other extreme, we have designers who have a vague idea of the group they are looking for. The queries that these designers pose will be far from capturing the input and output states or the functionality of the group they are searching for. For these designers their search can be characterized at best by still more abstract identifiers, i.e., more general keywords. These keywords can vary in terms of how specific they are. Obviously, general keywords can be associated with many groups, while more specialized keywords can prune down the possible set of candidate groups.

To accommodate both types of designers, a library structure is constructed that can be searched with identifiers at different levels of abstraction, the lowest level being that of specific group identifiers expressed as specifications of dynamic properties. The group library index is structured as a set of taxonomies (trees) of identifiers with isa relations between them. For more details of the generic approach behind this, see Section 5.1. The first tree contains identifiers at different levels of abstraction that describing functionality of the groups in the library. The root of the tree is the most general keyword for functionality. With each branching of the tree, the identifiers are specialized further. For example, in the middle of the tree an identifier such as reach_price_agreement can be used. We view the abstract group identifier (i.e., the dynamic group property specified from an external viewpoint) as the most specialized identifier for a group. Hence, the leaves of the tree correspond to individual abstract group identifiers.

The second tree contains the output information for the group. This may be useful for designers that have an idea of what output is to be used in the rest of the organization, but have no specific knowledge about functionalities. Again, at the root of this tree, the output state is described at the most general level. Going down the tree specializes the output. The leaves of the tree are the specific state properties based on output signatures that are also part of the abstract group identifier. The third tree has the same structure as the second tree, except that the nodes of the tree describe the inputs rather than the outputs.

Example 8. An example of such an output state property is then that there is a price over which X and Y agree.

∃ X, Y:ROLE ∃p communication_from_to(X, Y, inform, agreed_price_for(p, a)) &
communication_from_to(Y, X, inform, agreed_price_for(p, a))]

At the input then a similar property is that X and Y inform each other about a desire to agree on a price.

∃ X, Y:ROLE communication_from_to(X, Y, inform, desired_price_for(a)) & communication_from_to(Y, X, inform, desired_price_for(a))

4.2 Indexing by Environment Assumptions

The previous section describes how a designer can formulate a query for a desired group and get a set of groups that she could choose from. However, in many cases the designer can have additional constraints on the group. This subsection and the next ones classify important additional (meta)data about groups. This metadata can be supplied to the designer with the group, allowing her to investigate the properties of interest in more depth.

The first type of additional information about a group is formed by assumptions on their environment that guarantee conditions under which it can function properly. The environment of a group within an organization is formed by the rest of the organization and by the external world in which the organization is embedded. Assumptions may guarantee, for example, the availability of resources in the external world, or that upon certain requests generated as output by the group, other parts of the organization will provide answers as input for the group. An example of an environment assumption is:

Example 9. The following TTL expression captures that whenever a certain X has a request to a certain Y on a particular item (q), then outside the group a certain Y will somehow find this information from and communicate it to an X.

∀t [∃ X, Y:ROLE state(γ, t, output(G)) |= communication_from_to(X, Y, request, q) ⇒
∃t'≥t ∃a
∃ X, Y:ROLE state(γ, t', input(G)) |= communication_from_to(Y, X, inform, answer_for(a, q))]

In the previous example the information is received from outside group G. Then, for this group to function correctly, it should be used in an environment where the environment can satisfy the necessary information. This information could be generated by another group in the organization or from some other sources in the external world.

Environment assumptions can be addressed using a tree of different levels of abstraction similar to the cases shown in Section 4.1. From that perspective, the environment assumption given in Example 11 constitutes the leaves of the tree. These are the most specific nodes in the tree. The higher nodes of the tree would contain assumptions that are described in more general terms. Again here, as the higher nodes in the tree are again assumed to be generalizations of the lower nodes.

4.3 Indexing by Realization Constraints

The organization designer can also have constraints on how the organization will be realized. Most of these realization constraints are related to the agent allocation to the particular roles in the organization. That is, the designer can already have one or more agents that are going to take part in the organization. Obviously, an agent can play a role in the organization only if its properties are compatible with the properties of the role that it is going to play [Dastani et al., 2003]. Hence, a designer's choice of a group can also depend on the agents that are available.

This section discusses the criteria that need to be fulfilled to allocate agents to roles for the AGR approach. These criteria is crucial for realizing the organization dynamics successfully. One of the advantages of an organization model is that it abstracts from the specific agents fulfilling the roles. This means that all dynamic properties of the organization remain the same, independent of the particular allocated agents. However, the behaviors of these agents have to fulfill the dynamic properties of the roles and their interactions. The organization model can be (re)used for any allocation of agents to roles for which:

- for each role, the allocated agent's behavior satisfies the dynamic role properties,
- for each inter-group role interaction, one agent is allocated to both roles and its behavior satisfies the inter-group role interaction properties, and the communication between agents satisfies the respective transfer properties.

Given these requirements, a designer who already has a number of agents with particular behaviors can search the library with these agent behaviors. An agent behavior can be represented with a dynamic property, similar to the role properties, as shown in Example 12. Again, an agent behavior dynamic property is the most specific identifier and constitutes the leaves of a tree. On the other hand, a more general behavior description will appear on higher nodes of the tree.

Example 10. Let `communication_from_to(Role1, Role2, inform, desired_price_for(a))` denote that `Role1` is informing `Role2` that it is interested in agreeing on a price for item `a` and `communication_from_to(Role1, Role2, propose, price(p, a))` denote that `Role1` is proposing to `Role2` price `p` for item `a`. Then the following TTL formulation of an agent property means that whenever the agent A receives information of a desire for an agreed price from an agent B, then A will propose a price for the same item to B.

$$\forall t \; [\; \text{state}(\gamma, t, \text{input}(A)) \models \text{communication\_from\_to}(B, A, \text{inform}, \text{desired\_price\_for}(a))$$
$$\Rightarrow \exists t' \geq t \; \exists p \;\; \text{state}(\gamma, t', \text{output}(A)) \models \text{communication\_from\_to}(A, B, \text{propose}, \text{price}(p, a))$$

5 Querying the Library of Groups

Section 4 studies the different methods for indexing groups. In the section, we develop multi-dimensional library structure that combines the different indexing schemes as different dimensions. Using this multi-dimensional taxonomy, the designer can have interactions with the system to reformulate her queries. The system is in this sense interactive.

5.1 A Multi-Dimensional Taxonomic Approach

For the general approach it is just assumed that a number of taxonomies (the different contextual dimensions incorporated; cf. [Jonker and Vollebregt, 2000]) are given, where the nodes of these trees are used as identifiers for indexing the groups. For simplicity they are assumed to have the form of trees. Within these trees branches are defined by an isa-relation

`isa(n1, n2)` (or, in infix notation: `n1 isa n2`)

meaning that node `n2` is a direct specialization of node `n1` within one of the trees. Based on the indexing, each node n in one of the trees corresponds to a subset `gr(n)` of groups from the library, namely those indexed by node `n`. It is assumed that the different levels in the trees are abstraction levels: levels of specialization (going down) or generalization (going up). This means that if a group is indexed by a node n, then automatically it is considered that the nodes higher in the same tree apply to this group. This assumption implies that more specialized nodes correspond to smaller sets of groups:

$\mathtt{isa(n1,\ n2)} \Rightarrow \mathtt{gr(n1)} \subseteq \mathtt{gr(n2)}$

Moreover, a set S of nodes from possible different trees corresponds to the intersection of the sets of groups corresponding to the single nodes:

$\mathtt{gr(S)} = \bigcap_{n \in S} \mathtt{gr(n)}$

This general setup suggests two strategies to minimize the set of groups retrieved based on a query:

- within a tree, in the query try to use an identifier as low in the tree as possible (lower nodes provide smaller sets)
- use not just one node in a query but a set of nodes, taken from as many trees as possible (using more trees entails that the set of groups is made smaller since an intersection is made)

As designers may not be expected to express their queries in terms of nodes that are most appropriate, support can be offered to reformulate queries to more adequate ones. This is discussed in the next subsection.

5.2 Query Reformulation

First assume that the group library is queried using one identifier n. When this is the case, then the taxonomy of identifiers from which n is taken is used to aid the search. If the keyword searched already matches one of the leaves of the tree, then the set of groups associated with the leaf node is returned to the designer. Otherwise - if the query is matched to a node that is not a leaf - then the tree can be used to generate options for the user to further articulate her query. That is, starting from the node that matched the query, the designer can be asked to refine her query by proposing the branches of the tree as options. The underlying idea here is that the designer may know more about her needs than what she could initially formulate in a query. Hence, by posing choices to the designer, her query can be rephrased more precisely. Repeating the selection process will narrow the set of possible groups that will be returned for the query. If this query reformulation leads to one of the leaf nodes, then again the groups associated with that leaf

are returned. If the user gets stuck in choosing between two branches before reaching a leaf node, then all the groups below the current node are returned. For the case when more identifiers are present, the same can be done for other trees as well. This leads to a set of nodes S from the different trees for which each member shows a node that is as specialized as possible within the tree for that node. Then the groups corresponding to S is returned to the designer, which is the intersection of the sets of groups for all members of S.
In a case that a query leads to an empty set of groups, a reverse process of query reformulation may be needed: instead of specializing the query, in interaction with the designer it is generalized until the set of groups becomes non-empty. Finally, if an adequate query is reached, it is up to the designer to manually inspect the returned groups to choose an adequate group for her needs. Once the designer retrieves a group from a library, she can use it as it is or modify it further. Once, the group structure and the properties are finalized, the organization designer integrates the group into the organization structure. For our methodology, this would mean constructing appropriate group interaction properties between the new group and related existing groups.

Example 11. A designer's query could involve the identifier *payment* (for the tree describing the functionality dimension) as well as *price* (for the tree describing the output dimension). Both of these queries are vague; many groups could be related to payment as well as many groups could output some sort of price. By specializing both identifiers, a more specific combination of identifiers is reached, with a smaller set of groups.

5.3 Matching Query Terms and Indexing Terms

Given an abstract group identifier as a query, one needs to define a matching function that will be used to compare the query to the entries in the library. One matching function is exact matching which requires the terms of the query and the library entry to be exactly the same. A more loose way of matching is to the terms of the library entry to be logically stronger than that of the query. In other words, if an identifier of a group entails the term used in the query, one might consider it matching in a broader sense. Notice that such entailments can also be represented within the tree: a stronger term that entails a given identifier node within a tree can be added in this tree as a branch under this node. This addition can be left implicit if trees are not explicitly represented at forehand but (relevant parts of it) are generated during the process of using it.

Example 12. A query for a negotiation group is formulated where the output state is that the buyer and seller have reached an agreed price. Consider a group entry (Negotiate and Register) where in addition to reaching an agreed price, as an output it is also provided that the buyer and seller also register the price with a third party. With exact matching, the Negotiate-and-Register group will not be matched to the query. However, using entailment the group will be matched.

6 Discussion

Artikis *et al.* develop a framework to specify and animate computational societies[2002]. The specification of a society defines the social constraints, social roles, and social states. Social constraints define valid actions, permitted or prohibited actions, and the enforcement policies for these actions. A social role is defined as a set of preconditions and a set of constraints. The preconditions specify the requirements for an agent to play that role whereas the constraints specify what the agent should do once it is appointed to that role. Similar to our realization constraints, a role assignment procedure is then used to assign agents to roles based on the preconditions of the role and the capabilities of the agents as well as assignment constraints. A social state denotes the global state of a society based on the state of the environment, observable states of the involved agents, and states of the institutions. Using this framework, a society that carries out the contract-net protocol is formalized. Artikis *et al.* do not discuss a methodology for reusing their societies.

Padgham and Winikoff develop Promotheus, an agent-based software development methodology [2003]. This methodology is intended for non-experts and thus mostly defined informally. The methodology consists of system specification, architectural design, and detailed design phases. The system specification phase outlines the necessary functionality of the software. The architectural design phase divides the overall functionality into smaller tasks that will be carried out by different agents. Finally, the detailed design phase develops the individual agents that will carry out the tasks. Prometheus does not capture any functionality templates that can later be reused. In our approach, the abstraction of roles make it possible for different agents to play defined roles based on the realization constraints. More importantly, the abstraction of groups provides templates of functionality that can be stored in a library and be reused by other multiagent organizations.

Bussmann *et al.* identify a set of criteria to classify multiagent interaction protocols [2003]. Once the appropriate fields of these criteria are correctly set, the protocols can be classified and later be retrieved. Contrary to the criteria chosen in our approach, Bussmann et al. primarily consider quantitative properties, such as the number of agents involved, the number and size of the commitments between agents, and so on. Further, they do no consider a taxonomy of semantic identifiers as we have done here.

Malone *et al.* develop a library of business processes to help designers to create new organizations or restructure their existing organizations [1999]. The library stores processes in specialization hierarchies. A process entry includes the name and the description of the process as well as links to more general and more special processes. Similar to our approach, the process library is developed with human designers in mind. However, in our approach we formalize the groups as well as the queries to semi-automate the search. After some groups are retrieved from the library, then the designer can investigate them further and tailor one to her needs. The content and the size of Malone *et al.*'s process library is appealing for designing multiagent organizations. Our approach can benefit from starting with such a library and extending its entries with the formalized identifiers described before.

The formal approach presented in this paper provides a solid basis for a software environment supporting the interactive organization design process. Parts of this software environment are already available as separate components, in particular an editor to formally specify organization properties, group properties and role properties, and software to guide the query reformulation process, as initially developed in the ICEBERG project; for a survey of this project see [Jonker and Vollebregt, 2000]. In current research projects on organization modeling the software environment is being integrated and developed further.

References

Alexander Artikis, Jeremy Pitt, and Marek Sergot. Animated specifications of computational societies. In Proceedings of the 1st International Joint Conference on Autonomous Agents and MultiAgent Systems (AAMAS), pp. 1053-1061, 2002.

Stefan Bussmann, Nicholas R. Jennings, and Michael Wooldridge. Re-use of Interaction Protocols for Agent-based Control Applications. In: F. Giunchiglia et al. (eds.) Proceedings of the Workshop on Agent-Oriented Software Engineering (AOSE), Lecture Notes in Computer Science 2585, pp. 73-87, 2003.

Katleen M. Carley and Les Gasser. Computational Organization Theory in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Chapter 7. Gerhard Weiss, editor. MIT Press, 1999.

Mehdi Dastani, Virgina Dignum, and Frank Dignum. Role-Assignment in Open Agent Societies. In: Proceedings of the International Conference on Agents and MultiAgent Sysetms (AAMAS), pp.489-496, ACM Press, 2003.

Jacques Ferber and Olivier Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In: Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS), IEEE Computer Society Press, pp. 128-135, 1998.

Catholijn M. Jonker and Jan Treur. Relating Structure and Dynamics in an Organisation Model. In: J.S. Sichman, F. Bousquet, and P. Davidson (eds.), Multi-Agent-Based Simulation II, Proc. of the Third International Workshop on Multi-Agent Based Simulation, MABS'02. Lecture Notes in AI, vol. 2581, pp. 50-69, Springer Verlag, 2003.

Catholijn M. Jonker and Jan Treur. Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. International Journal of Cooperative Information Systems, vol. 11, pp. 51-92, 2002.

Catholijn M. Jonker and A. M. Vollebregt. ICEBERG: Exploiting Context in Information Brokering Agents. In: M. Klusch, L. Kerschberg (eds.), Cooperative Information Agents IV, Proceedings of the Fourth International Workshop on Cooperative Information Agents (CIA), Lecture Notes in Artificial Intelligence 1860, pp. 27-38, Springer Verlag, 2000.

Thomas W. Malone *et al.* Tools for Inventing Organizations: Toward a Handbook of Organizational Processes. In: Management Science, Vol. 45, No. 3, March 1999.

K. Meinke, J.V. Tucker (eds.), Many-Sorted Logic and Its Applications. Wiley and Sons, 1993.

Hafedh Mili, Ali Mili, Sherif Yacoub, and Edward Addy. Reuse-Based Software Engineering: Techniques, Organizations, and Controls. Jon Wiley & Sons, 2002.

Lin Padgham and Michael Winikoff. Prometheus: A Methodoloogy for Developing Intelligent Agents. In: F. Giunchiglia et al. (eds.) Proceedings of the Workshop on Agent-Oriented Software Engineering (AOSE), Lecture Notes in Computer Science 2585, pp. 174-185, 2003.

Hans van Vliet. Software Engineering: Principles and Practice. John Wiley and Sons, 2000.