Composition Norm Dynamics Calculation with Conceptual Graphs

Aldo de Moor

Infolab, Tilburg University, The Netherlands
e-mail: ademoor@kub.nl

Abstract. Network information system specification in virtual professional communities requires a legitimate user-driven approach. In such an approach, only specification changes are produced that are not only meaningful but also acceptable to all users. To do so, for each requested change, a relevant user group needs to be selected to work out the required knowledge definition changes. This paper describes the mechanism through which such a relevant user group can be calculated. The dynamics of the composition norms that guide the required specification behaviour are explained. The conceptual graph notation for four categories of specification knowledge is given. The Peirce conceptual graph workbench is used to demonstrate the composition norm dynamics calculation.

1 Introduction

Collaborative work is increasingly being done in a distributed fashion, supported by commonly available Internet-based information tools such as mailing lists or the web. We define the *virtual professional communities* in which such collaboration is to take place as communities or networks of professionals whose collaboration on activities required to realize shared goals is mostly or completely computer-enabled. The workflows of these communities are often supported by network information systems consisting of linked and configured standard information tools. The communal requirements and systems typically evolve strongly, with the users having an important role both as sources and as modellers of the system specifications. Active user participation in the specification process of such continuously evolving network information systems is very important, since community members have the most detailed knowledge about when breakdowns in work arise and how they can be resolved. One significant weakness of the traditional methods supporting specification processes is that they do not sufficiently involve the users (see [3] for a detailed study). They tend to rely on external analysts controlling the specification process, leaving the users only the rather passive role of being interviewed by them. Other methods, in particular socio-technical specification methods (such as Soft Systems Methodology), often overinvolve users in the sense of letting them participate in every conceivable change process. To increase the efficiency

[0] This paper has been published in Proceedings of the Eighth International Conference on Conceptual Structures - Logical, Linguistic, and Computational Issues (ICCS 2000), Darmstadt, Germany, August 2000. Lecture Notes in AI, No.1867, Springer-Verlag, Berlin, ISBN 3-540-67859-X, pp.522-535.

and willingness of users to participate in change processes, it must therefore be exactly known for each specification process what subset of all users is to take part, and in which capacity these are to be involved. To adequately determine the *relevant user group*, a *legitimate user-driven specification approach* is required. First, in such an approach, the community members are not just to provide specification knowledge, but also to control the process in which this knowledge is produced, thus making the specification proces truly *user-driven*. Second, the members of virtual professional communities, like their counterparts in traditional communities, are guided in their work by shared *social norms*. These norms should govern both the operations of a network and the specification processes in which the network and its information system is being defined. As these networks are egalitarian in nature, such norms cannot be imposed from above, but should originate from the community as a whole. Thus, the user-driven specification process needs to be *legitimate* as well, in the sense that specification changes are not only meaningful, but also *acceptable* to all members of the community. We call the norms that regulate the acceptable specification behaviour of the members of a virtual professional community *composition norms* (while we refer to the norms that regulate operational workflow behaviour as *action norms*).

The RENISYS (**RE**search **N**etwork **I**nformation **SY**stem **S**pecification) method [3] supports such a legitimate user-driven approach. It allows users facing a *breakdown* in their work to identify *problematic knowledge definitions* which they feel should be changed. For each of these definitions, RENISYS calculates the relevant user group, which it provides with the appropriate related knowledge definitions and the discussion environment needed for the group to work out the acceptable definition changes.

In [2], we explained how ontological and normative knowledge can be represented in conceptual graphs, and how these knowledge categories can be used to produce legitimate knowledge definition changes. In [4], we explored how the context lattices proposed in [8] can be applied to to efficiently structure, query, and update composition norms. Based on this work, we now show how the RENISYS method uses conceptual graph theory to determine the exact relevant user group required for a particular required specification change. To do so, the *composition norm dynamics* need to be calculated. In this way, a set of applicable norms can be calculated for each user and composition (part of the specification process necessary to resolve the breakdown). By then calculating what is the resultant effect of such a norm set, the method can determine whether a particular user is permitted, required, or prohibited to take part in certain stages of the specification process. This calculation, however, falls outside the scope of the current paper (see [3] for a detailed description).

Sect. 2 describes the semantics of composition norm dynamics. The conceptual graph notation of the various categories of knowledge definitions that are the output of specification processes are explained in Sect. 3. In Sect. 4, it is shown how the norm dynamics can be calculated using a standard conceptual graph workbench. The final section contains some discussion and conclusions.

2 Composition Norm Dynamics

The structure of composition norms has been extensively discussed elsewhere [3]. We therefore restrict ourselves here to briefly outline their main elements:

• *deontic effect*: the intended effect of a norm on the person who is to make a composition. A composition is either *permitted*, *required*, or *forbidden*.

• *actor*: the role, for example that of editor or reviewer, that a person is to play in order to be affected by the norm.

• *control process*. This concerns either an *initiation*, *execution*, or *evaluation* of the specification process at hand.

• *specification process*. In a specification process, a knowledge definition is changed. These change processes are either *creations*, *modifications*, or *terminations* of such definitions. The knowledge definitions themselves can be of four different categories: type definitions, state definitions, action norms, and composition norms. The role of the norms was already described in the previous section, while type definitions describe ontological knowledge, and state definitions represent states-of-affairs. For each knowledge definition category, a separate specification process has been defined. Thus, there are twelve customized specification processes, such as 'Create_Type' or 'Terminate_State'. The characteristics of these knowledge definitions and their specification processes have been discussed in detail in [4] and [3]. Their conceptual graph notation is presented in Sect. 3.

The formal notation for a composition norm d_{cn} is the following:
$d_{cn} = \langle id, de, a, cp, sp \rangle$, where id is the identifier of the norm, de is the deontic effect, a the actor, cp the control process and sp the specification process. An example of such a norm could be: $d_{cn_1} =$
$\langle \#12, Perm, List\_Owner, Exec, \underline{Modify\_Type(Mailing\_List)} \rangle$[1]. This norm says that a list owner is permitted to carry out changes in the (functionality) definitions of a mailing list, for example, by declaring the list to have an open instead of closed subscription procedure.

Example Assume that the set of legitimate composition norms $\mathcal{D}_{CN}$ consists of the following definitions:

- $\langle \#58, Perm, Publ\_Coord, Init, \underline{Terminate\_State(Reviewer)} \rangle$
- $\langle \#59, Req, Editor, Exec, \underline{Modify\_Type(Review)} \rangle$
- $\langle \#60, Perm, Actor, Control, \underline{Specify(Definition)} \rangle$
- $\langle \#61, Req, Editor, Control, \underline{Create\_Type(Edit)} \rangle$
- $\langle \#62, Forb, Journal\_Editor, \underline{Eval, Create\_Type(Edit)} \rangle$
- $\langle \#63, Req, Reviewer, Init, \underline{Create\_Type(Edit)} \rangle$
- $\langle \#64, Perm, Reviewer, Control, \underline{Create\_Type(Edit\_Report)} \rangle$
- $\langle \#65, Forb, Reviewer, Eval, \underline{Create\_Type(Edit\_Report)} \rangle$

[1] Note that, for simplicity, in the examples we represent the actor and control process entities by their types labels, instead of giving the full entity definition that would also include an identifier and a referent. Furthermore, the (nested) structure of the specification process is not yet formally defined, this we will do in Sect. 3. The predicate stands for the type of specification process, its argument for the knowledge definition being changed.

Composition norm #58 indicates that a publication coordinator may start the removal of a particular reviewer of a journal. Norm #59 expresses that an editor must revise the review process, if prompted. Norm #60 is a very generic norm, saying that any actor may control any specification process. Such a generic norm is typically defined at the initiation of a network, when the information system is still small in scope and only few users and actor roles have been defined. Norm #61 says that an editor must control the creation of new types of edit processes. However, according to norm #62 a journal editor is not allowed to evaluate such newly created process types. This norm could be introduced to ensure that such an editor cannot manipulate the results of his own work processes. Norm #63 says that a reviewer is responsible for starting the creation of a new edit process, for example when he is no longer satisfied with the way reviews are being handled. Norm #64 permits a reviewer to fully control the creation of report edit process types. Finally, norm #65 says that a reviewer is not allowed to evaluate a newly created report edit process definition. Such a privilege could instead be granted, for instance, only to the editorial board.

The types of the various norm elements are ordered using the following type hierarchy [2]:

T >
- Definition
- Entity >
  - PD_Actor >
    - *Editor* >
      - *Journal_Editor*
    - *Publ_Coord*
    - *Reviewer*
  - Control >
    - Init
    - Exec
    - Eval
  - Activity >
    - *Edit* >
      - *Edit_Report*
    - *Review*
  - Specify >
    - Create_Type
    - Modify_Type
    - Terminate_State

Composition norms play different roles depending on the users and specification processes they apply to at a particular moment in time. We refer to the way in which the status of norms can change as *norm dynamics*. These dynamics for composition norms can be summarized as follows: at any time, a composition norm base contains the set of *legitimate* norms $\mathcal{D}_{CN}$. A legitimate composition norm is *invoked* when there

[2] This hierarchy is formed by the relevant parts of the ontological framework introduced in [3], combined with some new, example-based types (in italics). Note that not all intermediate types are presented here to conserve space.

is at least one user with whom the norm matches. Invoked norms become *active* if they match with the *active specification process*, which is the process in which a problematic definition is currently to be changed. Each specification process consists of three parts: its initiation, execution, and evaluation. These parts are called the *specification process compositions*, which in the case of the active specification process we also refer to as *active compositions*. For each combination of user and active composition, a set of *applicable norms* exists, which determines what is the acceptable specification behaviour for that user and composition.

These norm dynamics need to be known, because they restrict the sets on which norm calculations need to be carried out. This is especially important in case of large numbers of norm definitions. In order to model the norm dynamics, two matching processes need to be defined. A *user match* is defined as a match between a user and an actor component of some composition norm. This means that at least one of the actor roles that the user plays is a subtype of the norm actor component. A *composition match* is defined as a match between a *specification process composition* and the composition part of some composition norm, also called the *norm composition part*. Such a match implies that the specification process composition must be a specialization of the norm composition part, as we support the view that generic norms are stronger than more specific norms. Thus, a specification process composition matches with, i.e. is governed by some composition norm, if the norm composition part is more generic than the specification process composition. In Sect. 4 we show how to calculate this match using conceptual graphs.

Definition 1
Let there be a composition norm $d_{cn} = \langle id, de, a, cp, sp \rangle \in \mathcal{D}_{CN}$. $\mathbf{comp}$ is a function on the argument of d_{cn} that produces the norm composition part: $\mathbf{comp}(d_{cn}) = \langle cp, sp \rangle$. Furthermore, $\mathcal{U}$ is the set of users, $\mathcal{E}$ is the set of entities ($\mathcal{U} \subset \mathcal{E}$). Function $\mathbf{type}(e)$ returns the type, and $\mathbf{ref}(e)$ returns the referent of entity e.

Let there be a user $u \in \mathcal{U}$ and an actor a of some composition norm $d_{cn} = \langle id, de, a, cp, sp \rangle \in \mathcal{D}_{CN}$. There is a *user match* between u and a, denoted as $u\ \theta_u\ a$, if $\exists\, e \in \mathcal{E} \mid \mathbf{ref}(e) = \mathbf{ref}(u) \wedge \mathbf{type}(e) \leq \mathbf{type}(a)$.

Let there be some specification process composition $comp \in \mathcal{C}omp$ (the set of all possible compositions, which is the Cartesian product of all control processes and all specification processes), and a norm composition part $\mathbf{comp}(d_{cn})$. There is a *composition match* between $comp$ and $\mathbf{comp}(d_{cn})$, denoted as $comp\ \theta_n\ \mathbf{comp}(d_{cn})$, if $comp$ is a specialization of $\mathbf{comp}(d_{cn})$. There is such a specialization if both the control process part and the specification process of $comp$ are specializations of their counterparts in $\mathbf{comp}(d_{cn})$. A specification process is a specialization of another specification process if both the type and the embedded knowledge definition of the first are specializations of the second.

Example The set of legitimate norms $\mathcal{D}_{CN} = \{\#58, \ldots, \#65\}$ The active specification process is the creation of a report edit process type, informally labeled $\underline{Create\_Type(Edit\_Report)}$. We further assume that the network contains two users: John, a journal editor, and Jack, a reviewer.

For composition norm #60, for instance, the norm composition part $\mathbf{comp}(\#60)$ is $\langle Control, \underline{Specify(Definition)}\rangle$. Given were a user John (more precisely: $u_1 \in \mathcal{U}$ with $\mathbf{type}(u_1) = User$ and $\mathbf{ref}(u_1) = John$)) and an entity e_1 (with $\mathbf{type}(e_1) = Journal\_Editor$ and $\mathbf{ref}(e_1) = John$)). Say there is an active composition $comp_1 = \langle Init, \underline{Create\_Type(Edit\_Report)}\rangle$:

$u_1\ \theta_u a$ holds, because $\mathbf{ref}(e_1) = \mathbf{ref}(u_1) \wedge \mathbf{type}(e_1) < Actor$ (the norm actor).

$comp_1\ \theta_n\ \mathbf{comp}(\#60)$ holds, because the various parts of $comp$ are specializations of their counterparts in $\mathbf{comp}(\#60)$.

We say that a (legitimate) composition norm becomes an *invoked composition norm* if there is a user match between some user and the norm actor.

Definition 2

D_{CN_I} is the set of invoked composition norms.

The function $\mathbf{D_{CN\_I}}$: $\mathcal{P}(\mathcal{U}) \rightarrow \mathcal{P}(\mathcal{D}_{CN})$ determines which legitimate composition norms are in D_{CN_I} (where $\mathcal{P}$ denotes the powerset):

$\mathbf{D_{CN\_I}}(\mathcal{U}) = \{d_{cn} = \langle id, de, a, cp, sp\rangle \in \mathcal{D}_{CN} \mid \exists\, u \in \mathcal{U} : u\ \theta_u\ a\}$

Example The set of invoked norms $D_{CN\_I} = \{\#59, \ldots, \#65\}$. Legitimate composition norm #58 is not invoked, because there are no users playing roles that are subtypes of the $Publ\_Coord$ actor role. All the other norms are invoked, because there is at least one user playing some role that is a subtype of these norms.

Whereas the invocation of legitimate norms depends on which users are participating in the community, the actual activation of the invoked composition norms depends on the currently *active specification process*. An invoked composition norm is an *active composition norm* if at least one of the active compositions (i.e. the initiation, execution, or evaluation of the active specification process) matches with the composition part of the invoked norm.

Definition 3

D_{CN_A} is the set of active composition norms.

sp_a is the active specification process. The set of active compositions $Comp_A = \{\langle Init, sp_a\rangle, \langle Exec, sp_a\rangle, \langle Eval, sp_a\rangle\}$.

The function $\mathbf{D_{CN\_A}} : \mathcal{SP} \rightarrow \mathcal{P}(D_{CN\_I})$ determines which invoked norms are in D_{CN_A}:

$\mathbf{D_{CN\_A}}(sp_a) = \{d_{cn\_i} \in D_{CN\_I} \mid \exists\, comp_a \in Comp_A : comp_a\ \theta_n\ \mathbf{comp}(d_{cn\_i})\}$

Example The set of active norms $D_{CN\_A} = \{\#60, \ldots, \#65\}$ For instance, none of the active compositions of the active specification process $Create\_Type(Edit\_Report)$ is a specialization of the norm composition part of invoked norm #59, which is therefore not an active composition norm. For all other invoked norms, there is at least one active composition of sp_a which is a specialization of the norm composition part.

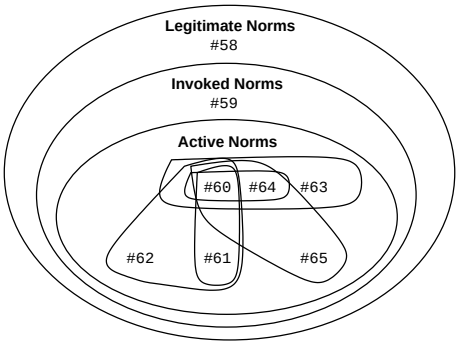

Fig. 1. Norm Dynamics Example

Active norms do not have an effect on all users. We call an active composition norm *applicable* to a particular user for a particular active composition if (1) the user matches with the actor part of the norm and (2) the active composition matches with the norm composition part.

Definition 4
$D_{CN\_APPL(u,comp_a)}$ is the set of applicable composition norms for user u and active composition $comp_a$. The function $\mathbf{D_{CN\_APPL}} : \mathcal{U} \times Comp_A \rightarrow \mathcal{P}(D_{CN\_A})$ determines which active norms are in $D_{CN\_APPL(u,comp_a)}$ for u and $comp_a$:
$\mathbf{D_{CN\_APPL}}(u, comp_a) =$
$\{d_{cn\_a} = \langle id, de, a, cp, sp\rangle \in D_{CN\_A} \mid u\ \theta_u\ a\ \wedge\ comp_a\ \theta_n\ \mathbf{comp}(d_{cn\_a})\}$

Example The applicable norm sets are:
- $D_{CN\_APPL(John,Init\_Create\_Type(Edit\_Report))} = \{\#60, \#61, \#62\}$
- $D_{CN\_APPL(John,Exec\_Create\_Type(Edit\_Report))} = \{\#60, \#61\}$
- $D_{CN\_APPL(John,Eval\_Create\_Type(Edit\_Report))} = \{\#60, \#61, \#62\}$
- $D_{CN\_APPL(Jack,Init\_Create\_Type(Edit\_Report))} = \{\#60, \#63, \#64\}$
- $D_{CN\_APPL(Jack,Exec\_Create\_Type(Edit\_Report))} = \{\#60, \#64\}$
- $D_{CN\_APPL(Jack,Eval\_Create\_Type(Edit\_Report))} = \{\#60, \#64, \#65\}$

For example, $D_{CN\_APPL(John,Init\_Create\_Type(Edit\_Report))}$ contains active composition norm #60, because (1) there is a user match between John and the actor component of this norm (the journal editor role that John plays is a subtype of $Actor$) and (2) there is a composition match between active composition $\langle Init, \underline{Create\_Type(Edit\_Report)}\rangle$ and norm composition part $\langle Control, \underline{Specify(Definition)}\rangle$.

The different norm sets are depicted in Fig. 1. The various subsets depicted within the set of active norms represent the different sets of applicable norms determined above. In the next section, the informal notation of the knowledge definitions that are the object of specification processes is made formal using conceptual graph notation.

3 Specification Knowledge Definition Representation

Four different categories of specification knowledge definitions are distinguished in RENISYS: type definitions, state definitions, action norms and composition norms. Type definitions are used to represent ontological knowledge, state definitions represent states-of-affairs, action norms regulate operational workflow behaviour, while composition norms govern the meta-level specification behaviour. Since the role of these definitions in the specification process has already been discussed quite extensively in [2] and [4], we here only show how they are represented in conceptual graphs, along with a simple example. One main advantage of using conceptual graphs over, for example, SQL tables and operations, is that in this way generalization hierarchies of specification knowledge can be taken into account.

The general graph structure of a specification knowledge definition is:

$[k : def]$.

Here, k is a knowledge category, whereas def represents the definition core in graph format. As the specific graph representation format of the definitions varies for the various knowledge categories, their formats are discussed separately.

3.1 Type Definitions

Definition 5
A type definition $d_t = \langle id, t_d, t_g, E, R\rangle \in \mathcal{D}_T$, with t_d the defined type, t_g the genus type, E a set of entities, and R the set of relations connecting them, is defined as:

$[\texttt{Type} : [t_d : *\texttt{x}] \rightarrow (\texttt{Def}) \rightarrow [t_g : *\texttt{x}]\ \mathbf{dif}(d_t)]$.

$\mathbf{dif}(\texttt{d}_\texttt{t})$, the differentia of the type definition, is connected to the genus concept $[\texttt{t}_\texttt{g} : *\texttt{x}]$ by its relations that have the genus placeholder X in their source or destination concepts. Thus, the differentia forms a subgraph that specializes the genus to the defined type. This representation of the type definition is different from the one used by Sowa (1984,p.106). The source entity of the (Def)-relation denotes the defined type, the destination entity the genus type.

Example A (partial) type definition of the report-editing process could be:

$[\texttt{Type} : [\texttt{Edit\_Report} : *\texttt{x}] \rightarrow (\texttt{Def}) \rightarrow [\texttt{Edit} : *\texttt{x}] -$
$(\texttt{Matr}) \rightarrow [\texttt{Draft\_Report}]$
$(\texttt{Rslt}) \rightarrow [\texttt{Edited\_Report}]]$.

3.2 State Definitions

Definition 6
A state definition $d_s = \langle id, E, R\rangle \in \mathcal{D}_S$ is defined as:

$[\texttt{State} : def]$.

def is the conceptual graph formed by the concepts in E linked by the relations in R. To be meaningful, state definitions need to be circumscribed by the available type

definitions.

Example The state definition that says that Harry is the list owner of the CG-mailing list is represented as:

$[\texttt{State} : [\texttt{List\_Owner} : \#\texttt{Harry}] \rightarrow (\texttt{Poss}) \rightarrow [\texttt{Mailing\_List} : \#\texttt{CG}]]$.

3.3 Action Norms

Definition 7

An action norm $d_{an} = \langle id, de, a, cp, w \rangle \in \mathcal{D}_{AN}$ is represented as follows:

$[an : a \leftarrow (\texttt{Agnt}) \leftarrow cp \rightarrow (\texttt{Obj}) \rightarrow w]$.

de is the deontic effect of the norm. a, cp, w stand for some actor[3], control process, and workflow, respectively. The norm category label an is:

$$an = \begin{cases} \texttt{Perm\_Act} & \text{if } de = Perm \\ \texttt{Req\_Act} & \text{if } de = Req \\ \texttt{Forb\_Act} & \text{if } de = Forb \end{cases}$$

Example The graph representation of the action norm that says that an editor may carry out the editing process is:

$[\texttt{Perm\_Act} : [\texttt{Editor}] \leftarrow (\texttt{Agnt}) \leftarrow [\texttt{Exec}] \rightarrow (\texttt{Obj}) \rightarrow [\texttt{Edit}]]$.

3.4 Composition Norms

Definition 8 A composition norm $d_{cn} = \langle id, de, a, cp, sp \rangle \in \mathcal{D}_{CN}$ is represented as:

$[cn : a \leftarrow (\texttt{Agnt}) \leftarrow cp \rightarrow (\texttt{Obj}) \rightarrow dp \rightarrow (\texttt{Rslt}) \rightarrow def]$.

Here, de, a, cp, sp mean the deontic effect, actor, control process, and specification process. $dp = [\mathbf{type}(\texttt{sp})]$ and $def = \mathbf{ref}(sp)$. The composition category label is:

$$cn = \begin{cases} \texttt{Perm\_Comp} & \text{if } de = Perm \\ \texttt{Req\_Comp} & \text{if } de = Req \\ \texttt{Forb\_Comp} & \text{if } de = Forb \end{cases}$$

Example Composition norm #58 is represented as:

$[\texttt{Perm\_Comp} : [\texttt{Publ\_Coord}] \leftarrow (\texttt{Agnt}) \leftarrow [\texttt{Init}] \rightarrow (\texttt{Obj}) -$
$[\texttt{Terminate\_State}] \rightarrow (\texttt{rslt}) \rightarrow [\texttt{State} : [\texttt{Reviewer}]]]$.

4 Norm Dynamics Calculation with Conceptual Graphs

This section illustrates how conceptual graph theory can be used to calculate the norm dynamics discussed in Sect. 2. To this purpose we use the Peirce conceptual graphs

[3] The meaning of the term *actor* is different from its interpretation in CGT, where it refers to a node in a dataflow graph that can perform computations on the declarative graph knowledge [9, p.188]

workbench[4], which was developed by Gerard Ellis. Among other things, the Peirce tool allows for the handling of nested graphs, which are needed to represent composition norms[5].

Type Hierarchy

The knowledge base has been loaded with the type hierarchy described in Sect. 2. These type definitions are represented as follows:

```
Editor < PD_Actor.
  Journal_Editor < Editor.
Publ_Coord < PD_Actor.
Reviewer < PD_Actor.
[...]
```

State Definitions

The Peirce knowledge base contains these state definitions to describe that user John is a journal editor and Jack a reviewer:

```
[State: [User: #John]].
[State: [User: #Jack]].
[State: [Journal_Editor: #John]].
[State: [Reviewer: #Jack]].
```

Composition Norms

The set $\mathcal{D}_{CN}$ consists of legitimate composition norms #58-#65. These norms are represented, in order, as:

```
[Perm_Comp: [Publ_Coord] <- (Agnt) <- [Init] -> (Obj) -
  -> [Terminate_State] -> (Rslt) -> [State: [Reviewer]]].
[Req_Comp: [Editor] <- (Agnt) <- [Exec] -> (Obj) -
  -> [Modify_Type] -> (Rslt) -> [Type: [Review]]].
[Perm_Comp: [Actor] <- (Agnt) <- [Control] -> (Obj) -
  -> [Specify] -> (Rslt) -> [Definition]].
[Req_Comp: [Editor] <- (Agnt) <- [Control] -> (Obj) -
  -> [Create_Type] -> (Rslt) -> [Type: [Edit]]].
[Forb_Comp: [Journal_Editor] <- (Agnt) <- [Eval] -> (Obj) -
  -> [Create_Type] -> (Rslt) -> [Type: [Edit]]].
[Req_Comp: [Reviewer] <- (Agnt) <- [Init] -> (Obj) -
  -> [Create_Type] -> (Rslt) -> [Type: [Edit]]].
[Perm_Comp: [Reviewer] <- (Agnt) <- [Control] -> (Obj) -
  -> [Create_Type] -> (Rslt) -> [Type: [Edit_Report]]].
[Forb_Comp: [Reviewer] <- (Agnt) <- [Eval] -> (Obj) -
  -> [Create_Type] -> (Rslt) -> [Type: [Edit_Report]]].
```

Invoked Norms Calculation

[4] http://www.cs.adelaide.edu.au/users/peirce/

[5] As in Peirce the symbols '@' and '.' cannot be used in the referent, they are both replaced by a '_'. The '>'-symbol indicates the prompt. A further explanation of the precise syntax of commands and graphs is given in [5] and is not repeated here.

For each composition norm $d_{cn} \in \mathcal{D}_{CN}$, d_{cn} is in the set of invoked composition norms D_{CN_I} if there is a user match $u\theta_u a$, with u being some user and a the actor part of d_{cn} (see Def.1 and 2).

The (temporary) set of user referents R_U consists of the referents of the user concepts in state definitions of users. These definitions are retrieved by the following operation.

```
> (Specialisations) -> [[State: [User]]]?
[State: [User: #John]].
[State: [User: #Jack]].
true
>
```

$R_U = \{\#John, \#Jack\}$

Now, $\forall d_{cn} \in \mathcal{D}_{CN}$, with a the type label of the actor part of d_{cn}: if $\exists\, r_u \in R_U$ such that there is a specialization of $[\texttt{State} : [a : r_u]]$, then $d_{cn} \in D_{CN\_I}$.

For example, for composition norm #58 (with $a = Publ\_Coord$):

```
> (Specialisations) -> [[State: [Publ_Coord: #John]]]?
no specializations
true
>
> (Specialisations) -> [[State: [Publ_Coord: #Jack]]]?
no specializations
true
>
```

Thus, composition norm #58 $\notin D_{CN\_I}$.

On the other hand, for composition norm #59 (with $a = Editor$):

```
> (Specialisations) -> [[State: [Editor: #John]]]?
[State: [Journal_Editor: #John]].
true
> (Specialisations) -> [[State: [Editor: #Jack]]]?
no specializations
true
```

Thus, since John is affected by it, composition norm #59 $\in D_{CN\_I}$. Similar calculations can be made for the other norms. It can thus be derived that D_{CN_I} consists of composition norms #59-#65.

Active Norms Calculation

An invoked norm is also an active norm if there is a composition match between one of the active compositions and the norm composition part (see Def.3).

In order to calculate the active norms, several temporary graphs are needed:

- Every active composition $comp_a \in Comp_A$ is stored in a separate graph.
- For each invoked composition norm $d_{cn\_i} \in D_{CN\_I}$, its norm composition part is stored in a new graph. Now, $\forall d_{cn\_i} \in D_{CN\_I} : d_{cn\_i} \in D_{CN\_A}$ if at least one of the active composition graphs is a specialization of the norm composition part graph. For example:
- The active composition graphs are:

```
[Init] -> (Obj) -> [Create_Type] -> (Rslt) -> [Type: [Edit_Report]].
[Exec] -> (Obj) -> [Create_Type] -> (Rslt) -> [Type: [Edit_Report]].
[Eval] -> (Obj) -> [Create_Type] -> (Rslt) -> [Type: [Edit_Report]].
```

• For invoked composition norm #59, the norm composition part graph is:

```
[Exec] -> (Obj) -> [Modify_Type] -> (Rslt) -> [Type: [Review]].
```

Performing the specialization operation gives the following result:

```
> (Specialisations) -> [[Exec] -> (Obj) -> [Modify_Type] -> (Rslt) -
  -> [Type: [Review]]]?
[Exec]->(Obj)->[Modify_Type]->(Rslt)->[Type: [Review]].
true
>
```

Since the operation only returns the norm composition graph itself, and none of the active composition graphs, composition norm #59 $\notin D_{CN\_A}$

• For invoked composition norm #60, however, the norm composition part graph is:

```
[Control] -> (Obj) -> [Specify] -> (Rslt) -> [Definition].
```

For this graph, the specialization operation returns:

```
> (Specialisations) -> [[Control] -> (Obj) -> [Specify] -> (Rslt) -
-> [Definition]]?
[Exec]->(Obj)->[Create_Type]->(Rslt)->[Type: [Edit_Report]].
[Init]->(Obj)->[Create_Type]->(Rslt)->[Type: [Edit_Report]].
[Control]->(Obj)->[Specify]->(Rslt)->[Definition].
[Eval]->(Obj)->[Create_Type]->(Rslt)->[Type: [Edit_Report]].
true
>
```

At least one (in fact, all three) of the active composition graphs are returned, so composition norm #60 $\in D_{CN\_A}$. Similar calculations can be made for the other norms. D_{CN_A} consists of composition norms #60-#65.

Applicable Norms Calculation

For each combination of user u and active composition $comp_a$, a set of applicable norms $D_{CN\_APPL(u,comp_a)}$ is defined (see Def.4). An active norm is in such a set if there are both a user match between u and the actor part of the norm, and a composition match between $comp_a$ and the norm composition part.

For example, to calculate $D_{CN\_APPL(u,comp_a)}$, with u = John and $comp_a = [\texttt{Init}] - > (\texttt{Obj}) - > [\texttt{Create\_Type}] - > (\texttt{Rslt}) - > [\texttt{Type} : [\texttt{Edit\_Report}]]$:

• To see whether composition norm #60, which has $Actor$ as the type label of its actor part, is in this set, we first need to determine whether there is a user match:

```
> (Specialisations) -> [[State: [Actor]]]?
[State: [User: #John]].
[State: [User: #Jack]].
[State: [Journal_Editor: #John]].
[State: [Reviewer: #Jack]].
true
>
```

Thus, there is indeed a user match between John and the actor part of norm #60. Now, it must be seen if there is a composition match as well.

The norm composition part graph for norm #60 is:

```
[Control] -> (Obj) -> [Specify] -> (Rslt) -> [Definition].
```

The matches with the active composition graphs are:

```
> (Specialisations) -> [[Control] -> (Obj) -> [Specify] ->
(Rslt) -> [Definition]]?
[Exec]->(Obj)->[Create_Type]->(Rslt)->[Type: [Edit_Report]].
[Init]->(Obj)->[Create_Type]->(Rslt)->[Type: [Edit_Report]].
[Control]->(Obj)->[Specify]->(Rslt)->[Definition].
[Eval]->(Obj)->[Create_Type]->(Rslt)->[Type: [Edit_Report]].
true
>
```

Thus, $comp_a$ is in the set of results, and the composition match is therefore successful. Since both the required user and composition match exist, composition norm #60 $\in D_{CN\_APPL(John,\ [Init]->(Obj)->[Create\_Type]->(Rslt)->[Type:\ [Edit\_Report]])}$.

Similar calculations can be made for the other composition norms in this set, as well as for the other applicable norm sets. A more efficient calculation would reuse the results of the user matches done in the calculation of the invoked norms, and the composition matches done for the active norms calculation. For clarity, the specialization operations were repeated here, however.

5 Discussion and Conclusions

Existing specification approaches are not very well-suited for network information system specification for virtual professional communities, since these require a legitimate user-driven approach. Traditional waterfall-based specification methods, such as SDM, are quite rigid and depend to a large extent on external analysts controlling the specification process [1]. Other methods, notably those based on a socio-technical paradigm, such as Soft Systems Methodology or ETHICS, assign a more prominent role to active user participation in the specification process [11, 6]. However, they still do not adequately support evolutionary systems development and are indiscriminate in which users to in involve in what particular specification change processes.

In this paper, we have demonstrated how to calculate the relevant group of users to involve in a particular specification change process. To this purpose, a user facing a breakdown in his work can identify *problematic knowledge definitions*, which he or she would like to see changed. Composition norms are essential to precisely regulate the specification processes needed to resolve these problematic definitions. They describe the meta-level change behaviour. This in contrast with numerous workflow modelling methods, either activity-based (i.e. specifying logistical workflows), or conversation-based (modelling communications and commitments) that do not provide guidelines on *who* is to change *what* [7]. In [3] we describe how to determine the *resultant deontic effect* of a set of applicable norms, which states whether a particular user is ultimately permitted, required, or forbidden to control (i.e. initiate, execute, or evaluate) a particular specification process. This, among other things, requires for occurring norm conflicts

to be resolved, which we have done making use of work done in dynamic deontic logic such as described in [12]. The actual change process is a form of a conversation by the selected users from the relevant user group. A Specification Process Model, based on Van Reijswoud's speech-act theory-based Transaction Process Model [10], prescribes the conversational moves that the various users can make. A prototype web server with mail functionality has been developed that can be used to support the specification process of a restricted set of knowledge definitions. Several case studies have been done that demonstrate how this support can facilitate network evolution. The still limited functionality of the tool will soon be upgraded to provide robust support for the full specification process.

Conceptual graph theory provides the theoretical constructs and tools to allow for such specification knowledge to be represented in a concise way and for the necessary calculations to be carried out efficiently. In [2], the importance of finding new applications such as these for CGT was discussed. We have now concretely demonstrated how existing tools such as the Peirce conceptual graph workbench can be applied to supporting the legitimate user-driven specification process. Of course, much work still needs to be done on optimizing the algorithms used, and on the integration of standard conceptual graph tool functionality with the RENISYS tool. These optimization and integration problems is the subject of current and future research.

References

1. F.P. Brooks. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, anniversary edition, 1995.
2. A. De Moor. Applying conceptual graph theory to the user-driven specification of network information systems. In *Proceedings of the Fifth International Conference on Conceptual Structures, University of Washington, Seattle, August 3–8, 1997*, pages 536–550. Springer-Verlag, 1997. Lecture Notes in Artificial Intelligence No. 1257.
3. A. De Moor. *Empowering the User: A Method for the Legitimate User-Driven Specification of Network Information Systems*. PhD thesis, Tilburg University, The Netherlands, 1999.
4. A. De Moor and G. Mineau. Handling specification knowledge evolution using context lattices. In *Proceedings of the Sixth International Conference on Conceptual Structures, ICCS'98, Montpellier, France, August 10–12, 1998*, pages 416–430, 1998.
5. G. Ellis and S. Callaghan. *PEIRCE User Manual*. Peirce Holdings International, Fitzroy, Australia, 1997.
6. R. Hirschheim and H.K. Klein. Realizing emancipatory principles in information systems development: The case for ETHICS. *Management Information Systems Quarterly*, 18(1):83–109, 1994.
7. M. Klein. Challenges and directions for coordination science. In *Proceedings of the Second International Conference on the Design of Cooperative Systems (COOP'96), Juan-les-Pins, France, June 12-14*, pages 705–722, 1996.
8. G. Mineau and O. Gerbé. Contexts: A formal definition of worlds of assertions. In *Proceedings of the Fifth International Conference on Conceptual Structures, University of Washington, Seattle, August 3–8, 1997*, pages 80–94. Springer Verlag, 1997. Lecture Notes in Artificial Intelligence, No. 1257.
9. J.F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.

10. V. Van Reijswoud. *The Structure of Business Communication: Theory, Model and Application*. PhD thesis, Delft University, 1996.
11. R. Vidgen. Stakeholders, soft systems and technology: Separation and mediation in the analysis of information system requirements. *Information Systems Journal*, 7:21–46, 1997.
12. R.J. Wieringa, J.-J.Ch. Meyer, and H. Weigand. Specifying dynamic and deontic integrity constraints. *Data and Knowledge Engineering*, 4:157–189, 1989.