Handling Specification Knowledge Evolution Using Context Lattices

Aldo de Moor[1] and Guy Mineau[2]

[1] Infolab, Tilburg University, P.O.Box 90153, 5000 LE Tilburg, The Netherlands
ademoor@kub.nl
[2] Université Laval, Department of Computer Science, Quebec City, Canada, G1K 7P4,
mineau@ift.ulaval.ca

Abstract. Internet-based information technologies have considerable potential for improving collaboration in professional communities. In this paper, we explain the concept of user-driven specification of network information systems that these communities require, and we describe some problems related to finding the right focus for adequate user involvement. A methodological approach to the management of specification knowledge definitions, which involves composition norms, is summarized. Subsequently, an existing conceptual graph based definitional framework for contexts is presented. Conceptual graphs are a simple, general, and powerful way of representing and reasoning about complex knowledge structures. This definitional framework organizes such graphs in context lattices, allowing for their efficient handling. We show how context lattices can be used for structuring composition norms. The approach makes use of context lattices in order to automatically verify specification constraints. To enable structured specification discourse, this mechanism is used to automatically select relevant users to be involved, as well as the information appropriate for building their discourse agendas. Consequently, this paper shows how conceptual graphs can play an important role in the development of this key Internet-based activity.

1 Introduction

Increasingly more distributed professional communities, such as research networks, are discovering the potential of collaboration through electronic media such as the Internet. However, several factors contribute to making it hard to determine the optimal or even just adequate use of information technology to support these networks in their collaborative activities [1]. One reason is that most knowledge creation activities are complex, situated, and dynamic. Another complicating factor is that numerous networked information tools are available, from which it is often difficult to determine which ones to use for what task purposes. Furthermore, system specification becomes even harder as it must also be user-driven, meaning that the users themselves are to discover 'breakdowns' in their use of the system and negotiate specification changes with other users and implementors. Users must initiate their own specification processes, because they

[0] This paper was published in the Proceedings of the Sixth International Conference on Conceptual Structures - Conceptual Structures: Theory, Tools, and Applications (ICCS'98), Montpellier, France, August 1998, pp.416-430. Lecture Notes in AI, No.1453, Springer-Verlag, Berlin.

themselves are the task experts, and moreover are often only loosely organized, without extensive organizational support for taking care of system development. For example, the publishing of electronic journals by networks of scholars instead of by commercial publishing houses is rapidly becoming popular. The support of the complex collaborative processes involved, such as the reviewing and editing of electronic publications, must not just be treated from a technical perspective. Rather, the new information tools must be designed to 'play an effective role within the social infrastructure of scholarship' [2]. The involved scholars themselves are in a good position to define this role, as they best understand the subtleties of their requirements, and can provide volunteer specification labour in these mostly underfunded joint projects.

To overcome the specification hurdles, structured methods for user-driven specification are needed. Already some approaches exist, for instance rapid application development, prototyping, and radically tailorable tools [3, 4], which more strongly involve users than traditional systems development methods. However, some drawbacks are that these approaches are based on traditional sequential instead of on evolutionary development models, focus too much on implementation rather than on conceptual issues, or support single user instead of group specification processes.

1.1 User-Driven Specification

True user-driven systems development means that each user can initiate and co-direct the specification process, based on concrete functionality problems he experiences when using the information system for his own purposes. Rather than doing a 'summative evaluation' of the information system in progress, in which users only approve of the overall specification process results, a user should be able to do a 'formative evaluation'. This entails that the users, rather than the developers, propose and decide upon specification suggestions which developers only help translate into actual modifications of the design of the system [5].

One approach that could in potential deal with the mentioned issues is process composition [6]. Its essence is that users of a system start with a rough definition of their work processes which are completely supported by the set of available tools. Over time, these specifications are gradually refined, always making sure that all processes are covered by available tool-enabled functionality. Such an approach takes into account the empirical findings that in general users initially only need to have an essential understanding of their business processes and tools to be able to initiate work [7], and that new technologies must be introduced gradually to prevent disruption of current work practices [8].

One implementation in progress of (group) process composition is the RENISYS specification method for research network information systems [1]. This method is discussed later on in this paper.

1.2 Finding a Focus

A major problem with process composition is that it is very difficult to determine the exact scope of a specification process aimed at resolving a functionality problem. Finding the proper scope is important in order to arrive at *legitimate* specifications, which

are not only meaningful, but also acceptable to the professional community as a whole [1]. However, this acceptability does not mean that all users should be consulted about every change all the time. Of course, on the one hand, all users who have an interest in the system component to be changed need to be involved. On the other hand, however, as few users as possible should participate in the resolution of a particular specification problem, in order to prevent 'specification overload', as well as to ensure the assignment of clear specification responsibilities.

Most current specification approaches intending to foster user participation do not systematically analyze how to achieve adequate user involvement in specification processes. For user participation in *specification discourse* (defined as rational discussion among users to reach agreement on the specifications of their network information system to become more satisfactory), it is at least necessary to precisely know:

1. When to consult users?
2. Which users to consult?
3. What to consult them about?
4. How to consult them?

Question 1 has to do with how to recognize *breakdowns*, which are disruptions in work processes experienced by participants while using the information system. A breakdown should trigger specification discourse resulting in newly defined functionality that better matches the real information needs of the user community. Question 4 focuses on how such specification discourse is to be systematically supported. Users could be provided with semi-structured linguistic options (representing for instance requests, assertions, promises), which are tailored to the particular specification problem at hand. Answering these two questions does not fall within the scope of the current paper. Ideas being worked out in the RENISYS project are taken from the language/action perspective [9]. This rather new paradigm for IS specification looks at the actions people carry out while communicating, and how this communication helps them to coordinate their activities. One of the key paradigmatic ideas is that people can make commitments as a result of speech acts. Such commitments in turn can be used to generate *agendas* of tasks to be carried out and evaluated by the various participating users. An agenda for a particular user thus consists of all things a user has to do, normally concerning the conduct and coordination of goal-oriented activities. In our case, however, agenda items refer to the specifications to be made or agreed upon of the network information system that supports the group work.

In this article, we will concentrate on questions 2 and 3. The main issues we will address are: (1) selecting the relevant users to participate in system specification discourse, and (2) determining the possibly different agendas for a particular specification discourse for the various selected users. We will do this by developing a mechanism to efficiently handle user-driven specification knowledge evolution using context lattices. These were first presented in [10], and will be briefly reintroduced in Sect. 3.3. The context lattices are used to (1) organize specification knowledge, (2) check whether knowledge definitions are legitimate (i.e. both meaningful and acceptable), and (3) determine which participants should be involved with what privileges in specification discourse to resolve illegitimate knowledge definitions. In Sect. 2, the approach to knowledge handling in the user-driven specification method RENISYS is described. Sect. 3 introduces

conceptual graph-based contexts and context lattices. In Sect. 4, context lattices are applied to structure what is called composition norm management, and in this way support the specification process.

2 Specification Knowledge Handling

First, the different categories of specification knowledge distinguished in the RENISYS method are presented. Then, the problem of how to ensure that specification changes are covered by what is called the composition norm closure, is discussed.

2.1 Knowledge Categories

RENISYS distinguishes three types of specification knowledge: ontological (type) definitions, state definitions, and norm definitions. The ontologies contain functionality specifications (what are the entities, attributes, and relationships to be represented and supported by the IS). States define what entities are or should be actually present. Norms determine (1) who can use the system (determined by action norms) and (2) who should be involved in their specification (determined by composition norms). Conceptual graphs are used as the underlying knowledge representation formalism because a knowledge representation formalism is needed that is sufficiently close to natural language to efficiently express complex specifications understandable to users, yet that is formal and constrained enough to allow for automated coordination of the specification process. CG theory is very well suited to this task, as argued in [1].

Type Definitions In RENISYS, the type definitions are organized into an ontological framework consisting of three kinds of ontologies. The heart of this framework is the core process ontology, consisting of elementary network process concepts derived from workflow modelling theory. Built on top of these generic concepts, three domain ontologies are defined. A domain is a system of network entities that can be observed by analyzing the universe of discourse from a particular perspective. The problem domain is the UoD seen from the task perspective, the human network is the UoD observed from the organizational perspective, and the information system is the same seen from the functionality perspective. The domain ontologies can be customized by the user to express concepts specific to his situation, thus allowing for conceptual evolution. Finally, the framework ontology describes a set of mapping constructs that link entities from the various domains.

Type definitions represent functionality specifications, such as the structure of documents, or the inputs and outputs of workflows. For example, a simplified definition of type MAILING_LIST could be:

```
[TYPE: [MAILING_LIST:*x] -> (def) -> [INFORMATION_TOOL:?x] -
   (matr) -> [RECEIVED_MAIL]
   (rslt) -> [RESENT_MAIL]
   (poss) <- [LIST_OWNER]].
```

Note that we do not use the standard type definition format introduced by Sowa, as we want a uniform representation format that can be used for all three categories of knowledge (i.e. types, norms, and states). Furthermore, we want to be able to represent and infer from qualified type definitions, such as partial, proposed, and invalid type definitions. For instance, partial type definitions must be identified and represented as such. They are incomplete definitions of the necessary properties that a concept type should have. They are very important in guiding specification discourse, as often a group of users will initially agree on a concept at least necessarily having a set of properties, while also agreeing that the definition is not yet complete. A partial type definition is thus open to further debate.

State Definitions State definitions represent states-of-affairs, which are first of all needed to determine which entities the information system implementation must support. For example, the following state definition indicates that John Doe is the list owner of the cg-mailing list. We thus know that all mailing list owner functions must be installed for at least this network participant.

```
[STATE: [MAILING_LIST: cg-list] <- (poss) <- [LIST_OWNER: John Doe]].
```

Also, state knowledge plays two crucial roles in the specification process of the network information system. First, it can be used to detect incomplete or inconsistent functionality specifications. For instance, if the type definition of a mailing list says that there should be at least one list owner, but (unlike in the above state definition) no such list owner has been defined, then a specification process can be started to specify who currently plays this role. Alternatively, if no such person can be defined, it may be that the type definition of mailing list must be revised so that this (currently mandatory) relation can be removed. Second, state definitions can be used as input objects into specification processes, for instance by allowing for the identification of subjects who can create new knowledge definitions. Such concrete assignments of specification responsibilities are essential for network information system development to be successful.

Norm Definitions Norm definitions represent deontic knowledge, which includes such concepts as responsibilities, permissions and prohibitions. This knowledge can, among other things, help to define and manage workflow and specification commitments. Formal models for such commitment management in a language/action context are dealt with in speech-act based deontic logic [11]. A key concept is that of actor, which is an interpreting entity capable of playing process controlling roles. Actor concepts themselves are ultimately instantiated by subjects, who are the people using and developing the network information system.

The basic pattern of a norm definition is an actor to which the norm applies, in combination with a control process (initiation, execution, or evaluation) and a transformation (a process in which a set of input objects is transformed into an output object) being controlled. Norm definitions can be subdivided into action norms and composition norms. An action is a control process plus the controlled (operational level) workflow, a composition is defined as a control process plus a (meta-level) specification process.

Action norms regulate behaviour at the operational level, in which case the transformations are called workflow processes. An example of an action norm is the following permitted action, which says that a list owner is permitted to add a list member:

```
[PERM_ACTION: [LIST_OWNER] <- (agnt) <- [EXEC] -> (obj) -
   [ADD_LIST_MEMBER]].
```

Composition norms, on the other hand, define desired behaviour at the specification level: they allow users who are, through actor roles involved in workflows, to be identified as simultaneously having legitimate roles in the specification process. Three kinds of specification processes are distinguished: creation, modification, and termination. An example of a composition norm could be this mandatory composition:

```
[MAND_COMP: [LIST_OWNER] <- (agnt) <- [EVAL] -> (obj) -
   [TERMINATE] -> (rslt) -> [TYPE: [LIST_MEMBER]].
```

The termination of a type means that a legitimate type is removed from the type hierarchy together with all its definitions, which may be required if a concept is no longer useful. This particular norm means that a list owner is required to evaluate (i.e. approve or reject) any list member type termination, which has been proposed by possibly another actor.

Having a well-supported approach for dealing with composition norms is crucial for managing the change process of network information systems. These norms help to identify which actors are to be involved in a particular specification process. Furthermore, they can be used to set the agenda for specification discourse, since they indicate what knowledge definitions an actor can legitimately handle and in what way. Thus, composition norms provide the key to answering the two questions we posed in section 1.2.

2.2 Composition Norm Closure

Traditional information systems analysis can be characterized as taking a snapshot of "the" sum of information requirements of an organization by a monolithic external group of analysts. However, in network information systems development, many users are often only temporarily involved in specification processes and this only from a very limited perspective and mandate: trying to resolve their own particular problem or that of others with whom they closely collaborate.

However, if every specification is linked to others and every specification must be covered by the appropriate composition norms, a major problem arises in case of (partially) changing needs: how to guarantee that proposed specification changes remain part of the *composition norm closure* (defined as the sum of the explicitly asserted plus all derivable composition norms), i.e., how to make sure that a proposed specification is legitimate and also does not leave any other specification uncovered?

To deal with this problem, it is often not enough to find just one applicable norm. Completeness is very important. For instance, if one wants to know whether the current user, who plays a number of actor roles, is allowed to change the definition of a particular type, all composition norms applicable to this definition need to be identified.

However, as the knowledge base of graphs grows large, checking every unorganized composition norm by standard projections can get very cumbersome. This is especially true when recursive operations on embedded parts must be carried out. Furthermore, such a straightforward approach does not easily generate related contextual information, such as the other definitions the actor specifying the current definition is allowed to make.

Therefore, a more sophisticated norm querying and updating mechanism is needed. Such a query mechanism, which is optimized to handle particular contexts and the relations between different worlds of assertions, is formed by context lattices. Two of the major advantages of context lattices are that they (1) allow queries to be simplified, as embedded queries can be subdivided into their constituting parts and (2) the structure of the knowledge base can be queried, allowing for interesting relations to be easily discovered [10].

3 Contexts

Composition norms play a crucial role in the coordination of the user-driven specification process, as they put constraints on who is authorized to (re)define which particular knowledge definitions. Thus, the knowledge definitions are only true if the specification process conditions under which they are asserted are true as well. Such conditional sheets of assertion can be naturally represented as conceptual graph contexts [10].

Contexts are an essential building block of conceptual graph theory [12]. Building on these notions, Mineau and Gerbé [1997] presented a formal theory of context lattices, which is briefly summarized here.

A context is a conceptual device that can be used for organizing information that originates from multiple worlds of assertion. It consists of an extension and an intention. In a context, the truth of a set of assertions (the extension) depends on a specific set of conditions (the intention). Thus, the intention is formed by those graphs which, if conjunctively satisfied, make the extension true. Thus, only if the intention graphs can all be made true, do the extension graphs exist. A context C_i is defined as a tuple of two sets of conceptual graphs:

$$C_i =< T, G > \tag{1}$$

where T is the intention, and G is the extension of C_i. Two functions I and E were defined so that for a context C_i its intention T equals $I(C_i)$ and the extension G is the same as $E(C_i)$.

Contexts can directly be used to represent norms. The intention of a (composition) norm defines that some actor is capable of controlling a specification process of some kind of knowledge definition. The graph representation of this most generic composition norm intention is:

```
[ACTOR] <- (agnt) <- [CONTROL] -> (obj) -> [SPECIFY] -
   (rslt) -> [DEFINITION: #]
```

It will be used as the intention of some context, while the referent of the DEFINITION concept, representing the knowledge definition being specified, will be considered as being in the extension of the same context. The format of the extension graph

depends on the type of this definition (i.e. TYPE, PERM_ACTION, or STATE). Examples of these definitions were given in the previous section.

3.1 Example: Mailing List

We will illustrate the ideas put forward in this paper with a short example of a specification process typically encountered in a research network. To clarify the ideas introduced in this paper, only three *permitted* compositions are given. In a realistic case, required and forbidden compositions will also be needed.

The example is the following. Many research networks are supported by mailing lists. A mailing list comes installed with a default set of properties. Some public lists allow any member to control the change of all their properties, which is explicitly defined as a composition norm. Often, however, as the networks grow in scope, the mailing list is to play new roles. For example, the purpose for which the mailing list is used could be changed from enabling general information exchange to supporting the preparation of a confidential report. In case of such a private mailing list, the list owner, who is a special type of network actor, can explicitly be allowed to modify the setting of the list parameters. Finally, (for any type of mailing list) a list owner can start the cancellation of the action norm which says that a list applicant can register himself as a list member.

In this case, the following three (permitted) composition norms apply:

(1) In a mailing list, any network participant is permitted to control (initiate, execute, and evaluate) modifications of mailing list properties, for example when the scope of the group needs to be changed.

(2) In case of a private mailing list, a list owner is permitted to make modifications of the properties of the mailing list, i.e. he may change the settings about whether the list has open or closed subscription, whether it is moderated or not, etc.

(3) A list owner is allowed to initiate the termination of the action (norm) that a list applicant can register himself as a list member.

As contexts C_i, these composition norms could be represented as follows:

```
-------------------------------------------------------------------------
| C1: Perm_Comp_1                                                       |
| i1: [ACTOR] <- (agnt) <- [CONTROL] -> (obj) -> [MODIFY] -             |
|       (rslt) -> [TYPE: #]                                             |
-------------------------------------------------------------------------
| g1: [MAILING_LIST]                                                    |
-------------------------------------------------------------------------

-------------------------------------------------------------------------
| C2: Perm_Comp_2                                                       |
| i2: [LIST_OWNER] <- (agnt) <- [EXEC] -> (obj) -> [MODIFY] -           |
|       (rslt) -> [TYPE: #]                                             |
-------------------------------------------------------------------------
| g2: [PRIVATE_MAILING_LIST]                                            |
-------------------------------------------------------------------------

-------------------------------------------------------------------------
| C3: Perm_Comp_3                                                       |
| i3: [LIST_OWNER] <- (agnt) <- [INIT] -> (obj) -> [TERMINATE] -        |
|       (rslt) -> [PERM_ACTION: #]                                      |
-------------------------------------------------------------------------
| g3: [LIST_APPLICANT] <- (agnt) <- [EXEC] -> (obj) -> [REG_LIST_MEMBER]  |
-------------------------------------------------------------------------
```

Hereby we assume that this (partial) type hierarchy has been defined:

```
[T] >
  [CONTROL] >
    [EXEC]
    [INIT]
  [DEFINITION] >
    [PERM_ACTION]
    [TYPE]
  [INFO_TOOL] >
    [MAILING_LIST] >
      [PRIVATE_MAILING_LIST]
  [ACTOR] >
    [LIST_APPLICANT]
    [LIST_OWNER]
  [REG_LIST_MEMBER]
  [SPECIFY] >
    [MODIFY]
    [TERMINATE]
```

The contexts thus allow for a clear separation between the knowledge being specified (g_i), and the modality of the actual specification process (i_i). Note that contexts instead of non-nested CGs are not only used in C3, but in C1 and C2 as well, because the intention (actor permitted to control specification process) represents the conditions under which the extension (knowledge definition, e.g. mailing list) may be specified.

3.2 Basic Context Inferences

Contexts have some interesting properties that can be inferred from the previous definitions [10].

First, it is important to realize that a graph g can be in $E(C_i)$ either because it has been explicitly asserted in that context, or because it is part of the transitive closure of the asserted graphs of that context. Thus, if g_m is asserted in some context but is a generalization of g_n from $E(C_i)$, then g_m is also considered to be in $E(C_i)$. Thus, in the example, $E(C_2)$ contains both g_2 and g_1, as a MAILING_LIST is a generalization of a PRIVATE_MAILING_LIST.

Second, if $I(C_j) < I(C_i)$ then $E(C_j) \subseteq E(C_i)$. This means that if the intention of C_j is a specialization of the intention of C_i, then (at least) all the extension graphs of C_j are in the extension of C_i. Thus, as $i_2 < i_1$, we can derive that $E(C_1) = \{g_1, g_2\}$. Note that the contexts described in Sect. 3.1 only contain the extension graphs that have been *explicitly* asserted. Later on in this paper we will also include the *derived* extension graphs.

Now that we have made some basic context inferences, we will look at how they can be used in the construction of context lattices.

3.3 Context Lattices

A context lattice is a structure that can be used to organize a set of contexts, allowing associations between these contexts to be made.

A context lattice L consists of a set of *formal contexts* C_i^*, which are structured in a partial order $\leq$.

A formal context C_i^* is represented as:

$$C_i^* =< I^*(G_i), E^*(T_i) > \quad (2)$$

where $I^*(G_i) = \cup_j I(C_j) | G_i \subseteq E(C_j)$
and $E^*(T_i) = \cap_j E(C_j) | I(C_j) \subseteq T_i$

Context lattices can be used to optimize query mechanisms concerning (1) particular contexts and (2) the relations between sets of intentions and extensions. In other words, they can be used to make explicit the relations between different worlds of assertions [10].

To create the context lattice for our example, we need to take the following steps:

1. Recalculate the contexts

As noted earlier, contexts can have explicitly asserted as well as derived extension graphs. The representations of C_1, C_2 and C_3 only showed the explicitly asserted extension graphs:

$C_1 =< \{i_1\}, \{g_1\} >$
$C_2 =< \{i_2\}, \{g_2\} >$
$C_3 =< \{i_3\}, \{g_3\} >$

With completely calculated extensions (using the inferences of section 3.2 to recalculate $E(C_i)$) the contexts can be represented as:

$C_1 =< \{i_1\}, \{g_1, g_2\} >$
$C_2 =< \{i_2\}, \{g_1, g_2\} >$
$C_3 =< \{i_3\}, \{g_3\} >$

Note that C_1 and C_2 now have the same set of extension graphs.

2. Calculate the formal contexts

For each context, we now calculate the formal context (using $C_i^* =< I^*(G_i), E^*(T_i) >$).

$C_1^* =< \{i_1, i_2\}, \{g_1, g_2\} >$
$C_2^* =< \{i_1, i_2\}, \{g_1, g_2\} >$
$C_3^* =< \{i_3\}, \{g_3\} >$

For C_1 and C_2 the same formal context is produced.

3. Calculate the context lattice

Each formal context should occur only once, so redundant formal contexts (i.e. the above C_2^*) must be removed. Furthermore, in order to create a lattice, we must also add a formal context including all extension graphs, as well as a context including all intention graphs. After renumbering the formal contexts, the resulting context lattice is as represented in fig. 1:

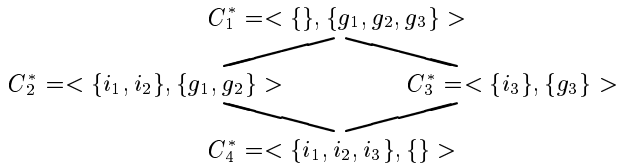

Fig. 1. The Context Lattice for the Example

4 Structuring Composition Norm Management

On top of the context lattice structure, a mechanism that allows for its efficient querying has been defined [10]. One of its main advantages is that complex queries consisting of sequences of steps and involving both intentions and extensions can be formulated. This mechanism can be used for a structured yet flexible approach to norm management in specification discourse, by automatically determining which users to involve in discussions about specification and changes and what they are to discuss about (their agendas).

There are two main ways in which a context lattice can be used in a user-driven specification process. First, it can be used to assess whether a new knowledge definition is legitimate by checking if a particular specification process is covered by some composition norm. As this is a relatively simple task of projecting the specification process on the composition norm base, we do not work out this application here.

The second application of context lattices in the specification process is applying (new) specification constraints (constraints on the relations that hold between different knowledge definitions) on the existing (type, norm, and state) knowledge bases. This differs from the first application as, after a constraint has been applied, originally legitimate knowledge base definitions may become illegitimate, and would then need to be respecified.

In this section we will first give a brief summary of how context lattices can be queried. Then, it will be illustrated how this query mechanism can play a role in composition norm management, by applying it to our example in the resolution of one realistic specification constraint.

4.1 Querying Concept Lattices

In order to make series of consecutive queries where the result of one query is the input for the embedding query, which is needed for navigating a context lattice, we need two more constructs. First, we need to be able to query a particular context extension or intention. Second, we must be able to identify the context which matches the result of an extension or intention-directed query.

For the first purpose, two query functions have been defined that allow respectively extension or intention graphs to be retrieved from a specified formal context [10]:

$$\delta_{E^*}(C^*, q) = \{g \epsilon E(C^*) | g \leq q\} \tag{3}$$

$$\delta_{I^*}(C^*, q) = \{g \epsilon I(C^*) | g \leq q\} \tag{4}$$

Furthermore, Mineau and Gerbé have constructed two context-identifying functions:

$$C_E(G) =< I^*(G), E^*(I^*(G)) > \tag{5}$$

$$C_I(T) =< I^*(E^*(T)), E^*(T) > \tag{6}$$

Space does not permit to describe the inner workings of these functions in detail (see [10] for further explanation). Right now, it suffices to understand that these functions allow the most specific context related to respectively a set of extension graphs G

or a set of intention graphs T to be found Together, these functions can be used to produce *embedded queries* by alternately querying and identifying contexts, thus enabling navigation through the context lattice.

4.2 Supporting the Specification Process

In the applications of context lattices, discussed in the previous section, the following general steps apply:

1) Check either the specification of a new knowledge definition against the composition norm base or the specifications of an existing knowledge base against a specification constraint.

2) Identify the resulting illegitimate knowledge definition(s).

3) Identify appropriate 'remedial composition norms' (i.e. composition norms in which the illegitimate knowledge definition is in the extension)

4) Build discourse agendas (overviews of the specifications to discuss) for the users identified by those remedial composition norms, so that they can start resolving the illegitimate definitions.

These processes consist of sequences of queries that switch their focus between what is being defined and who is defining. For this purpose, the functions provided by context lattice theory are concise and powerful, at least from a conceptual point of view.

One way in which we can apply context lattices is by formulating *specification constraints*, which constrain possible specifications and can be expressed as (sequences of) composition norm queries. Note that the example of the resolution of a specification constraint presented next is simple, and the translation into context lattice queries is not yet very elegant. However, what we try to present here is the general idea that flattening queries using context lattices is a powerful tool for simplifying and helping to understand queries with respect to the contexts where they apply. In future work, we aim to develop a more standardized approach that can apply to different situations.

4.3 An Example

One specification constraint could be:

"Only actors involved in the definition of permitted actions are to be involved in the definition of (the functionality of) information tools".

The constraint guarantees that enabling technical functionality is defined only by those who are also involved in defining the use that is being made of at least some of these tools. This specification constraint and much more complex ones can be helpful to realize more user-driven specification, tailored to the unique characteristics of a professional community. The power of the approach developed in this paper is that it allows such constraints to be easily checked against any existing norm base, identifying (now) illegitimate knowledge definitions, and providing the contextual information necessary for their resolution.

We will illustrate these rather abstract notions by translating the above mentioned informal specification constraint into a concrete sequence of composition norm queries.

Decomposing the specification constraint, we must answer the following questions:

1. Which actors control the specification of which information tools?
2. Are there illegitimate composition norms (because some of these norm actors are not also being involved in the specification of any permitted actions?)
3. Which actors are to respecify these illegitimate norms on the basis of what agendas?

Questions **1-3** can be decomposed into the following steps (this decomposition is not trivial, in future research we aim at providing guidelines to achieve it):

1a. Determine which specializations g_j of information tools have been defined. The query s_1 should start at the top of the context lattice, as this context contains all extension graphs:

$s_1 = \delta_{E^*}(C_1^*, q_1) = \{g_1, g_2\} =$ {[MAILING_LIST],[PRIVATE_MAILING_LIST]}

where $q_1 =$ [INFO_TOOL]

1b. For each of these information tools g_j, determine which actors a_i control its specification:

$s_2 = \delta_{I^*}(C_E(g_1), q_2) = \delta_{I^*}(C_2^*, q_2) = \{i_1, i_2\}$
$s_3 = \delta_{I^*}(C_E(g_2), q_2) = \delta_{I^*}(C_2^*, q_2) = \{i_1, i_2\}$
where $q_2 =$ [ACTOR:?] <- (agnt) <- [CONTROL] -> (obj) -> [SPECIFY] -
(rslt) -> [TYPE]
$a_2 =$ [ACTOR:?] = {[ACTOR],[LIST_OWNER]}

and $a_3 =$ [ACTOR:?] = {[ACTOR],[LIST_OWNER]}

2a. Determine which actors a_i are involved in the specification of permitted actions. This query should be directed toward the bottom of the context lattice, as this context contains all intention graphs (which in turn include the desired actor concepts):

$s_4 = \delta_{I^*}(C_4^*, q_4) = \{i_3\}$
where $q_4 =$ [ACTOR:?] <- (agnt) <- [CONTROL] -> (obj) -> [SPECIFY] -
(rslt) -> [PERM_ACTION]

$a_4 =$ [ACTOR:?] = {[LIST_OWNER]}

2b. Using a_4, determine, for each type of information tool g_j (see **1a**), its corresponding s_i and actors a_i(see **1b**), which actors $a_i^{'}$ currently illegitimately control its specification process.

g_1: [MAILING_LIST]
$a_2^{'} = a_2 - (a_2 \cap a_4) =$ {[ACTOR]}
g_2: [PRIVATE_MAILING_LIST]
$a_3^{'} = a_3 - (a_3 \cap a_4) =$ {[ACTOR]}

2c. For each tool identified by the g_j having illegitimate controlling actors $a_i^{'}$, define the illegitimate composition norms $c_k^{'} =< i_l, g_j >$ by selecting from the s_i from **1b** those i_l which contain $a_i^{'}$:

$c_1^{'} =< i_1, g_1 >$
$c_2^{'} =< i_1, g_2 >$

3. In the previous two steps we identified the illegitimate norms. Now we will prepare the stage for the specification discourse in which these norms are to be corrected. A composition norm does not just need to be seen as a context. It is itself a knowledge definition which needs to be covered by the extension graph of at least one other composition norm, which in that case acts as a meta-norm. In order to correct the illegitimate norms we need to (a) identify which actors are permitted to do this and (b) what items should be on their specification agenda. This step falls outside the scope of this paper but is presented here to provide the reader with the whole picture. A forthcoming paper will elaborate on meta-norms and contexts of contexts.

3a. For each illegitimate composition norm $c_k^{'}$, select the actors a_i from the permitted (meta) composition norms c_m which allow that $c_k^{'}$ to be modified:[1]

$c_m =< i_m, g_m >$
where i_m = `[ACTOR:?] <- (agnt) <- [EXEC] -> (obj) -> [MODIFY] -> (rslt) -> [PERM_COMP: #]`

and $g_m = c_k^{'}$

3b. For each of these actors a_i, build an agenda A_i. Such an agenda could consist of (1) all illegitimate norms $c_k^{'}$ that each actor is permitted to respecify and (2) contextual information from the most specific context in which these norms are represented, or other contexts which are related to this context in some significant way.

The exact contextual graphs to be included in these agendas are determined by the way in which the specification discourse is being supported, which is not covered in this paper and needs considerable future research. However, we would like to give some idea of the direction we are investigating. In our example, we identified the illegitimate (derived) composition norm 'any actor is permitted to control (i.e. initiate, execute, and evaluate) the specification of a private mailing list' ($< i_1, g_2 >$). From its formal context C_2^* it also appears that a list owner, on the other hand, is permitted to at least execute the modification of this type ($< i_2, g_2 >$). If another specification constraint would say that one permitted composition for each control process category per knowledge definition suffices, then only the initiation and evaluation of the modification now would remain to be defined (as the execution of the modification of the private mailing list type is already covered by the norm referring to the list owner). Thus, the specification agendas A_i for the actors a_i identified in **3a** could include : 'you can be involved in the respecification of the initiation and the evaluation of the modification of the type private mailing list', as well as 'there is also actor-such-and-such (e.g. the list owner) who has the same (or more general/specific) specification rights, with whom you can negotiate or whom you can ask for advice.'. Of course, in a well-supported discourse these kinds of agendas would be translated into statements and queries much more readable to their human interpreters, but such issues are of a linguistic nature and are not dealt with here.

5 Conclusions

Rapid change in work practices and supporting information technology is becoming an ever more important aspect of life in many distributed professional communities. One of their critical success factors therefore is the continuous involvement of users in the (re)specification of their network information system.

In this paper, the conceptual graph-based approach for the navigation of context lattices developed by Mineau and Gerbé [1997] was used to structure the handling of user-driven specification knowledge evolution. In virtual professional communities, the various kinds of norms and the knowledge definitions to which they apply, as well as the specification constraints that apply to these norms, are prone to change. The formal context lattice approach can be used to guarantee that specification processes result in

[1] For lack of space, we have not included such composition norms in our example, but since they are also represented in a context lattice, the same mechanisms apply. The only difference is that the extension graphs are themselves contexts (as defined in Sect.3).

legitimate knowledge definitions, which are both meaningful and acceptable to the user community. Extracting the context to which a query is applied, provides simpler graphs that can more easily be understood by the user when he interacts with the CG base. It also provides a hierarchical path that guides the matching process between CGs, that would otherwise not be there to guide the search. Even though the computation cost of matching graphs would be the same, overall performance would be improved by these guidelines as the search is more constrained. But the most interesting part about using a context lattice, is that it provides a structuring of different contexts that help conceptualize (and possibly visualize) how different contexts ('micro-worlds') relate to one another, adding to the conceptualization power of conceptual graphs.

In future research, we plan to further formalize and standardize the still quite conceptual approach presented here, and also look into issues regarding its implementation.

References

1. 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.
2. B.R. Gaines. Dimensions of electronic journals. In T.M. Harrison and T. Stephen, editors, *Computer Networking and Scholarly Communication in the Twenty-First Century*, pages 315–339. State University of New York Press, 1996.
3. L.J. Arthur. *Rapid Evolutionary Development - Requirements, Prototyping & Software Creation*. John Wiley & Sons, 1992.
4. T.W. Malone, K.-Y. Lai, and C. Fry. Experiments with Oval: A radically tailorable tool for cooperative work. *ACM Transactions on Information Systems*, 13(2):177–205, 1995.
5. P. Holt. User-centred design and writing tools: Designing with writers, not for writers. *Intelligent Tutoring Media*, 3(2/3):53–63, 1992.
6. G. Fitzpatrick and J. Welsh. Process support: Inflexible imposition or chaotic composition? *Interacting with Computers*, 7(2):167–180, 1995.
7. L.J. Arthur. Quantum improvements in software system quality. *Communications of the ACM*, 40(6):46–52, 1997.
8. I. Hawryszkiewycz. A framework for strategic planning for communications support. In *Proceedings of the Inaugural Conference of Informatics in Multinational Enterprises, Washington, October 1997*, 1997.
9. F. Dignum, J. Dietz, E. Verharen, and H. Weigand, editors. *Proceedings of the First International Workshop on Communication Modeling 'Communication Modeling - The Language/Action Perspective', Tilburg, The Netherlands, July 1-2, 1996*. Springer eWiC series, 1996. http://www.springer.co.uk/eWiC/Workshops/CM96.html.
10. 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.
11. F. Dignum and H. Weigand. Communication and deontic logic. In R. Wieringa and R. Feenstra, editors, *Working Papers of the IS-CORE Workshop on Information Systems - Correctness and Reusability, Amsterdam, 26-30 September, 1994*, pages 401–415, September 1994.
12. J.F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.