Making Virtual Communities Work: Matching Their Functionalities

Aldo de Moor and Willem-Jan van den Heuvel

Infolab
Tilburg University
P.O. Box 90153, 5000 LE Tilburg, The Netherlands
ademoor/wjheuvel@kub.nl

Abstract. Virtual professional communities increasingly make use of standard information tools, like mailers and groupware applications, to support their collaborative activities. However, the requirements of these communities and the technologies in use change rapidly, so that requirements and available functionalities continuously need to be recalibrated. Changing their mappings is not trivial, because of the many dependencies between the business processes and tool components. To increase the efficiency of the specification process, functionality matching approaches need to be developed that are sensitive to the socio-technical semantics of the community. In this way, the technical feasibility of a proposed change can be more easily determined.
In this paper, we propose a concrete matching approach based on the RENISYS method for legitimate user-driven system specification. The approach consists of a series of matching process steps which are based on a functionality matching meta-model. We illustrate how such an approach could be used in practice by applying it to a proposed system change process in the case of an electronic journal[1].

1 Introduction

Virtual professional communities and their information systems are good examples of complex socio-technical systems. There is significant pressure on these systems to change, because of change drivers of many different kinds. Technological, economic, political and many other factors contribute to a continuous need for evolution of the requirements and supporting information technologies. However, change processes are costly, and effects of changes are often unclear. Therefore, often considerable resistance to change exists. To reduce this resistance, it must be clear to users what are the consequences of a proposed change in the socio-technical system. An important barrier is taken away if changes are legitimate, in the sense that they are both meaningful and acceptable to the community. One approach increasing this legitimacy is the RENISYS method [5]. Other effects of change, such as those on non-functional constraints like quality and usability aspects, need to be taken into account as well.

[1] Published in: Proceedings of the Ninth International Conference on Conceptual Structures (ICCS 2001), Stanford University, USA, July 30 - August 3, 2001. Lecture Notes in Computer Science, No.2120, Springer-Verlag, Berlin pp.260-274.

Yet another very important category of change aspects is ensuring a good match between functional requirements and the available IT resources. The question ”do we still have adequate technological support after implementing the proposed change?” needs to be answered positively, especially since in virtual communities work processes are complely or mostly enabled by information technologies. Otherwise, disruption of the socio-technical infrastructure will interrupt the evolution of the community. Furthermore, when the technical complexity of specification changes can be reduced, then the efficiency of the change process can be increased so that more attention can be paid to other, non-functional aspects. This will lead to information systems that are better tailored to the specific needs of the community.

In Sect. 2, we first give an overview of existing theory and practice concerning functionality matching, and introduce a case to illustrate the ideas. In Sect. 3, we then introduce a meta-model specifically developed for matching required and enabled functionalities in virtual professional communities. This meta-model is based on the RENISYS method. A concrete matching process, grounded in this meta-model is introduced in Sect. 4. Some conclusions and directions for future research are given in Sect. 5.

2 Functionality Matching: Theory and Practice

In this section, we first define our view on functionality matching. After reviewing related work, we introduce a case that is used to explain the ideas introduced in this paper.

2.1 Theory: What is Functionality Matching?

Information systems for virtual professional communities are generally not constructed from scratch. Instead, applications supporting collaboration are developed by experimenting with widely available *information tools*, which originally were often developed for other purposes [7]. We define an information tool as a unit of software that completely or partially enables some information and communication processes. An *information process* allows a single user to produce a new *information object* out of already existing objects. An example is a researcher writing a review of a paper. The focus of a *communication process*, which involves multiple communicating entities, is on the transfer rather than on the production of information objects. Examples of information tools range from mailers, list servers, and chat tools to numerous kinds of web applications.

In order to understand the role that an information tool plays as part of the socio-technical network information system, we need to look at both the *functionality* that the tool provides and its *usability*, which concerns the extent to which the functions provided by the tool are understood and applied by its users to their particular tasks. Together, these notions determine the *effective functionality* [9], which we define as that part of the available functionality used to support the activities of the community. This needs to be known to assess the effect of changes in the information system.

Usability is not a property of the tool itself, but rather of the tool in its context of use. Therefore, usability has been defined as the evaluation of the extent to which users

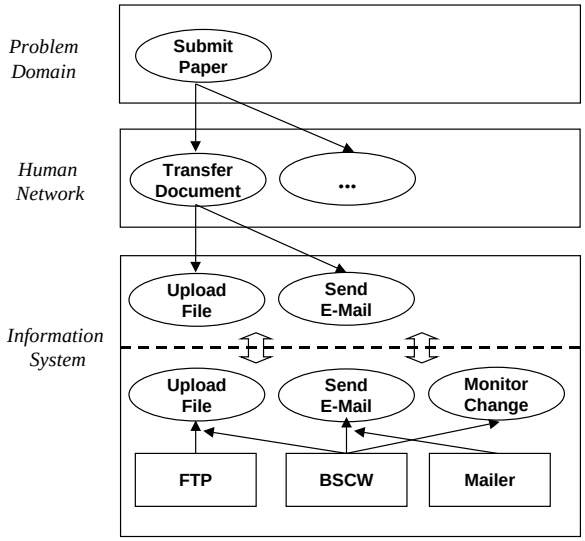

Fig. 1. Functionality Specification Example: The Paper Submission Process

can translate their intentions into effective actions to access the functionality [8]. We have decomposed this definition to focus on two aspects: (1) who can *access* a particular information tool in which capacity and (2) how to *represent* the user requirements, the tool functionality and their linkages. To deal with the first aspect, some framework is needed to model the functionality of tool types and the access rights of particular users to particular instances of these tools. The second aspect requires some ontology describing the key entities of the complete socio-technical system, in order for users to propose and discuss about functionality specifications. In the RENISYS method, both aspects are addressed by the *reference framework* that is used to represent specifications that define the socio-technical system. The reference framework consists of a *problem domain* ontology modelling goals and activities, a *human network* ontology representing the organizational structures in which these tasks are carried out, and an *information system* domain ontology in which the technical functionality used by the virtual community for its work is defined. An example of the dependencies between elements from these domains is shown in Fig. 1. The relationships between the particular elements of the figure are explained in Sect. 2.3.

2.2 Related Theory and Approaches

To some extent, research into matching algorithms that compare the functionality of two software components on the basis of some kind of component specification, has been done in the areas of information retrieval [15], cooperating (or interoperable) information systems ([13], [3]) and software reusability ([14], [12]). These solutions assume that the functionality of components can be represented as a collection of signatures.

A component signature explicitly separates the definition of the services of the component from the actual implementation. The services are defined as methods (functions) with input parameters, input types and the output type. This separation is critical for interoperability across programming languages, operating systems and even networks. The Interface Definition Language (IDL) is a prominent example of a interface specification language, that has been proposed by the Object Management Group (OMG) and constitutes the foundation for their object request broker (middleware) architecture. IDL specifications can be used to specify component attributes, parent classes, typed events, methods (including input and output parameters and their data types), and exceptions.

In the following excerpt we give a simple example of an IDL specification:

```
module MyCommunity {
 interface Administrator : Person {
   attribute integer ID;
   void registerNewMember (in short MemberID, in integer ID,
          in String Community) raises (NotAuthorized);
   void deregisterMember (in short MemberID, in integer ID,
          in String Community) raises (NotAuthorized);}
} /* End MyCommunity
```

The excerpt specifies an interface of a `Administrator` component. This class inherits the characteristics and behavior from the parent class `Person`. `Administrator` has an attribute `ID` with `integer` as its datatype. Moreover, this class exhibits three methods to other interested classes: `registerNewMember`, `deregisterMember` and `NotAuthorized`. The exception `NotAuthorized` occurs whenever the person that tries to invoke one of these methods is not authorized.

Most interface matching approaches now compare the methods, and pre and post-conditions of a collection of interface specifications, that are stored in some kind of interface repository, with a given specification. The solutions generally have some mechanism to deal with partial matches, and result in the best matching interface specification(s).

Although these ideas are applicable for acquiring potentially reusable component definitions for example from a component repository, they do not deal with the specific functionality evolution characteristics of virtual professional communities. More particularly, such socio-technical systems require efficient mechanisms to deal with changes to configurations of tools, requirements, and users. Besides that, current matching approaches only match functionality in the narrow sense, omitting the usability aspect.

In our view, mapping tool functionality to the requirements of virtual communities, requires firstly a functionality specification language that adds more social-technical system semantics to the rather low-level interface definitions, and secondly, a mapping procedure that is based on a more sophisticated process that makes use of these semantics.

This does not mean that component mapping is unnecessary. On the contrary, these approaches are essential to *construct* the support information tool components, e.g., mailing component, chat enabling components and registration components for composing virtual community applications. However, they are not capable of dealing with the more complex, and high level information tool requirement specifications specified by the (mostly non-technical) community members themselves. Questions like ”do

we still have enabling components if we change the community structure?" can not be answered with interface mapping approaches as the specification languages can not capture the semantics.

Thus, what is needed are approaches that can deal with the specific functionality matching problems of virtual communities, so that changes in functionality can be analyzed in their broader usage context.

2.3 Case: The Electronic Journal on Comparative Law

IWI, a Dutch organization stimulating new ways of distributing scientific information, funded a project to create an Electronic Journal on Comparative Law (EJCL)[2]. The project group included participants from various academic law institutes, university libraries, and computer centers. The goal was to have all publishing activities, ranging from paper submission to editing, peer review, and publication, being done completely via the Web. The initial basic set of requirements defined by the project team members gradually grew in scope and complexity. Furthermore, the set of simple information tools over time included more advanced groupware applications.

One interesting observation from a functionality matching perspective concerns the definition of the technological support for the paper submission process (Fig. 1). The submission of papers was considered as a document transferring process, which consisted of two required communication processes: first, an author has to upload a file, then he sends an e-mail to the editor with the submission details. The technical committee responsible for the selection of the right tools proposed to enable the file uploading process using a standard FTP tool. This tool enables basic file transfer. However, the project coordinator then proposed to use a BSCW-server instead. This tool has been optimized for file distribution processes, as it enables advanced, userfriendly, and secure file transfer. Furthermore, it can be used to send e-mails as well as monitor changes in file updates and accesses. The effects of *replacing* the FTP-server with a BSCW-server are not clear. Both tools enable their own sets of information and communication (IC) processes. Their effective functionality needs to be known before this change is technically feasible. The approach we introduce next is capable of dealing with such change complexities.

3 A Functionality Matching Meta-Model

To develop an approach that can facilitate the functionality change process, we first need to define a functionality matching meta-model. This metamodel can be used to model the exact relations between tools, users, and the functionalities that are required and enabled. We use this static model to define the actual functionality matching *process* in Sect. 4.

Before presenting the meta-model, we first operationalize the concept of effective functionality by listing a number of axioms.

[2] http://law.kub.nl/ejcl

Effective Tool Functionality Axioms These axioms form the foundation of the functionality matching meta-model. In any change process, their validity must be guaranteed.

- An information tool can enable one or more information and communication processes.
  Example: a mailer allows a user to compose a mail (information process) and send or receive a mail (communication processes).
- Different information tools may have partially *overlapping* functionality, i.e. each enabling the same information or communication process, while also enabling different such processes at the same time.
  Example: Both a mailer and a web browser allow one to send a mail. However, only with a mailer can a user also organize sent and received messages, whereas sophisticated HTML document access is just possible with a web browser.
- All network participants involved in a required information/communication process must have at least one *enabling information tool* at their disposal.
  Example: a participant may have a required communication process of sending a mail. Thus, the participant must have access to, for instance, a mailer or a web browser.

The Meta-Model In the meta-model, we describe how in RENISYS the following elements are specified: (1) the enabled functionality (which tools enable which IC-processes), (2) the required functionality (which IC-processes are required), (3) the enabling functionality (which required IC-processes can be enabled by the tools), (4) functionality access (which users have access to which tool instances), and (5) functionality assignment (which users use which tool instances for what workflow mappings). Fig. 2 shows the relations between the different entities necessary in the functionality specification process[3]. The semantics of this figure are explained in the remainder of this section.

Enabled Functionality Any IC-process enabled by some information tool is called an *enabled IC-process*. Such a process is represented as a a state definition which conforms to a specialization of the following type definition of the enable-relation:

$$[\texttt{Type} : [\texttt{Enable} : *\texttt{x}] \rightarrow (\texttt{Def}) \rightarrow [\texttt{T} : *\texttt{x}]-$$
$$(\texttt{Inst}) \rightarrow [\texttt{Info\_Tool}]$$
$$(\texttt{Obj}) \rightarrow [\texttt{IC\_Proc}]].$$

Example

The following state definition says that uploading a file is enabled by an FTP tool:

$$[\texttt{State} : [\texttt{Enable} : \#267]-$$
$$(\texttt{Inst}) \rightarrow [\texttt{FTP}]$$
$$(\texttt{Obj}) \rightarrow [\texttt{Upload\_File}]].$$

□

[3] The diagram is a variant of NIAM-notation [6]. Bold arrows indicate subtype relations, the predicates represent other relations. Only the entity types User, Info_Tool, IC_Process, and Workflow_Mapping are basic concept types. The other entities distinguished in the functionality specification process are roles that these types play. They are denoted by an asterix.

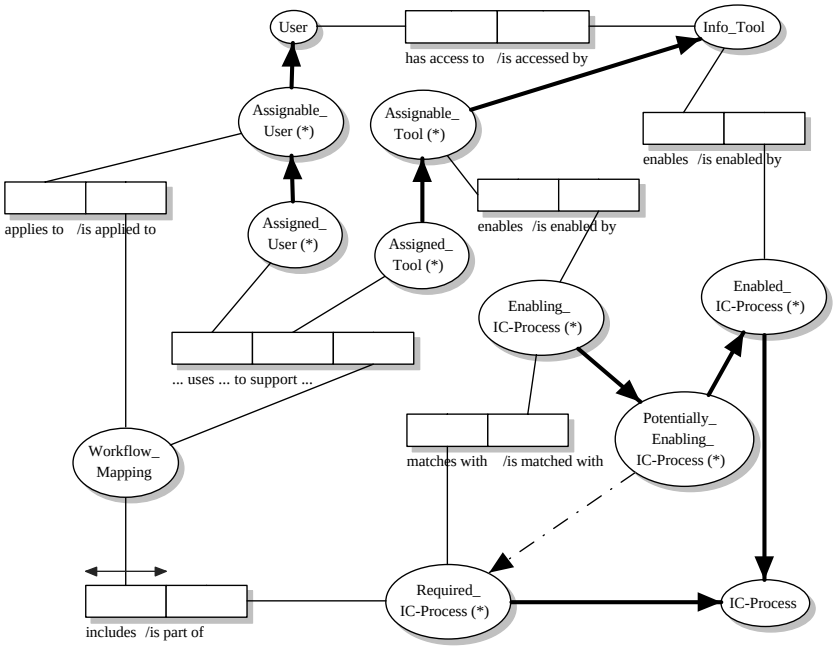

Fig. 2. A Functionality Matching Meta-Model

Required Functionality The RENISYS reference framework distinguishes three domains, as mentioned before. Workflows from the problem domain are called *activities*, from the human network *interactions* and from the information system domain *IC-processes*. Functionality requirements consist of information or communication (IC) processes in their usage context. Requirements are represented by workflow mappings, which relate a workflow from the problem domain, via one in the human network domain to a workflow in the information system domain. For example, a workflow mapping can say that a (problem domain) editorial process is a form of a (human network) discussion process which is supported by an (information system) file sending process, among others. The *required IC-process* then is the IC-process part of the workflow mapping, in this case the *send file*-process. The activity and interaction part of such a mapping together identify the usage context in which the required IC-process operates. A particular workflow mapping is represented as a state definition which conforms to a specialization of the workflow mapping type definition. This definition is:

$$
\begin{aligned}
&[\texttt{Type} : [\texttt{Workflow\_Mapping} : *\texttt{x}] \rightarrow (\texttt{Def}) \rightarrow [\texttt{Mapping} : *\texttt{x}]-\\
&\quad (\texttt{Part}) \rightarrow [\texttt{Activity}]\\
&\quad (\texttt{Part}) \rightarrow [\texttt{Interaction}]
\end{aligned}
$$

(Part) → [IC_Proc]].

Example

[State : [Workflow_Mapping : #123]–
(Part) → [Submit_Paper]
(Part) → [Transfer_Document]
(Part) → [Comm_Process]].

This workflow mapping specifies that 'the *paper submission* process is a *document transfer* process that is supported by some *communication* process'. The latter process is thus a required IC-process. Note that this process is defined as a generic communication process, because the specifier does not either know, or care, by which particular type of communication process the paper submission process is to be supported. This means that many degrees of freedom are left in the choice of the tools that are to support this particular workflow mapping.

□

Enabling Functionality For the required IC-process of each workflow mapping, a set of *potentially enabling IC-processes* exists. These are those IC-processes that are (1) enabled by some tool and (2) are a subtype of the required IC-process. This makes sense, because the specifiers of a workflow mapping would define the required IC-process to be generic if they are indifferent or do not know yet which particular enabling IC-process should satisfy it, as in the previous example. So, the more generic the required IC-process, the more enabled IC-processes can match with it, thus the more potentially enabling IC-processes for a particular workflow mapping there are. Out of this set of potentially enabling IC-processes, at least one *enabling IC-process* must be selected for the workflow mapping to become operational.

Example

Assume the workflow mapping #123 defined earlier, and assume that the set of enabled IC-processes equals
{[Send_Mail], [Receive_Mail], [Send_File], [Edit_Textfile]}, of which all but the edit-textfile process (which is an information process) are communication processes. The set of potentially enabling IC-processes is
{[Send_Mail], [Receive_Mail], [Send_File]}, since these are all subtypes of the *communication*-process. Out of this set of potentially enabling processes, the specifier selects the *send mail*-process as the (actually) *enabling IC-process*.

□

Functionality Access Each user has *access* to a certain set of *tool instances*, represented in the form of state definitions that conform to this type definition of the access-relation:

[Type : [Access : *x] → (Def) → [T : *x]–
(Poss) ← [User]
(Obj) → [Info_Tool]].

Example

The following state definition (representing a state-of-affairs in the domain) indicates that John has access to mailer #4 at the Infolab.

$$[\texttt{State} : [\texttt{Access} : \#213]-\\ (\texttt{Poss}) \leftarrow [\texttt{User} : \#\texttt{John}]\\ (\texttt{Obj}) \rightarrow [\texttt{Mailer} : \#4@\texttt{Infolab}]].$$

□

Some types of information tools are *complex*, in the sense that users can access only part of the functionality of the tool. A typical example of such a complex information tool is a web server that consists of many different pages, each enabling different functionality.

The meaning of a complex information tool is the following:

$$[\texttt{Type} : [\texttt{Complex\_Info\_Tool} : *\texttt{x}] \rightarrow (\texttt{Def}) \rightarrow [\texttt{Info\_Tool} : *\texttt{x}]-\\ (\texttt{Part}) \rightarrow [\texttt{Entity}]].$$

Example

This definition of a complex information tool indicates that user John only has access to the home page of the BCFOR-web server:

$$[\texttt{State} : [\texttt{Access} : \#215]-\\ (\texttt{Poss}) \leftarrow [\texttt{User} : \#\texttt{John}]\\ (\texttt{Obj}) \rightarrow [\texttt{Web\_Server} : \#\texttt{BCFOR}] \rightarrow (\texttt{Part}) \rightarrow [\texttt{Web\_Page} : \#\texttt{home.html}]].$$

□

Functionality Assignment For each workflow mapping, it should be determined for all users in the community whether the workflow mapping applies to them. If so, out of the tools accessible to a particular user, one or more should be selected. This selected tool is to support him in the required IC-process that is part of the workflow mapping.

Users are in the set of *assignable users*, a subset of all community members, for a workflow mapping if he or she is permitted to be involved in it. Such permissions can in principle be calculated from the action norms that define the workflow behaviour of users (see [5]), however, for simplicity, we allow users to be assigned to a workflow mapping manually here.

An information tool is in the set of *assignable tools* for a workflow mapping if it enables the enabling IC-process, i.e., the particular IC-process type chosen to match with the required IC-process. Thus, assignable tools for the *send mail* required IC-process of the previous example could be, for instance, mailers and BSCW, since both tools offer some form of mail-sending functionality.

The actual assignment of the tool that is to support a particular assignable user in a specific workflow mapping is not automated in our approach. The main reason is that the choice of which tool to use for a work process depends on many circumstances beyond functionality matching, such as the non-functional requirements mentioned in the introduction. For example, the users themselves could be intensively involved in this assignment process, since they can best assess their own requirements and preferences.

The functionality assignment is represented by a so-called *support*-relation. This definition assigns some assignable user and an assignable tool to the workflow mapping. This user is referred to as the *assigned user*, the tool is called the *assigned tool*. The type definition of the support-relation is:

```
[Type : [Support : *x] → (Def) → [T : *x]–
   (Poss) ← [User]
   (Inst) → [Info_Tool]
   (Obj) → [Workflow_Mapping]].
```

Example

The requirement that user John is to use (possibly among other tools) BSCW server #3 to enable him to submit papers is represented by:

```
[State : [Support : #167]–
   (Poss) ← [User : #John]
   (Inst) → [BSCW : #3]
   (Obj) → [Workflow_Mapping : #123]].
```

□

Often, it may be necessary to specify that a particular required IC-process is to be supported by a particular type of tool, without knowing yet who are its users. For example, a team leader wants all of his staff to use the same tool for a particular workflow. The representation of such a *required implementation* is:

```
[Type : [Req_Impl : *x] → (Def) → [Entity : *x]–
   (Inst) → [Info_Tool]
   (Obj) → [Workflow_Mapping]].
```

Example

The following state definition concisely represents that all users should be able to access the BSCW-server for submitting papers:

```
[State : [Req_Impl : #165]–
   (Inst) → [Web_Server : #BSCW]
   (Obj) → [Workflow_Mapping : #123]].
```

□

4 The Functionality Matching Process

In the previous section, we introduced the static meta-model in which the matching relations between requirements (i.e. workflow mappings), tools, and users were specified. Next, we use this model to construct a process that can be used to assess the effects of changes in the system specifications on the match between required and enabled functionalities. We briefly introduce the matching steps in Sect. 4.1, and we illustrate the process using material from the case described earlier in Sect. 2.3.

4.1 Matching Process Steps

The matching process consists of 5 stages: (1) creating a base of system specifications, (2) proposing some change to the specifications, (3) formulating a set of functionality matching criteria, (4) calculating the match, and (5) interpreting the results.

1. Define System Specifications Virtual professional communities are continuously changing socio-technical systems. At a time t=0, before the change is proposed, we assume that the current system specifications are properly matched with respect to the required and enabled functionalities.

Example A set of enabled-functionality state definitions declares that FTP enables the upload file-process; BSCW enables the upload file, send e-mail and monitor change-process; mailers enable the send e-mail process. To model the required functionality, two workflow mapping definitions WM1 and WM2 represent that the submit paper-process is a transfer document-process which is enabled by the upload file-process (WM1) and the send e-mail-process(WM2), respectively. For WM1, the only potentially enabling IC-process is the upload file-process (enabled by FTP and BSCW), for WM2 the only potentially enabling IC-process is send e-mail (enabled by BSCW and mailers). These are also selected as the enabling IC-processes for the workflow mappings. The selection process is trivial in this case, since there is only one potentially enabling IC-process here. In other cases, however, there may be more options to choose from, if there are deeper IC-process type hierarchies. Two functionality access definitions declare that John has access to mailer #4 and FTP-server #EJCL, three other definitions say that Mary has access to mailer #22, FTP-server #EJCL and BSCW-server #EJCL. Finally, to assign the functionality, two support definitions declare that John is supported in WM1 by FTP-server #EJCL and in WM2 by mailer #4. Two other definitions say that Mary is supported in WM1 by FTP-server #EJCL and in WM2 by mailer #22.

These definitions are represented in the Peirce[4] conceptual graph workbench. To illustrate, one of these definitions is given here:

```
[State: [Support:#300] -
   <- (Poss) <- [User:#John]
   -> (Inst) -> [FTP:#EJCL]
   -> (Obj) -> [Workflow_Mapping:#WM1]].
```

2. Propose Specification Change At t=1, some specification change is proposed by one of the users. Such a change concerns the creation, modification or deletion of one or more specification knowledge definitions like the ones presented above. Note that the legitimacy of the user being involved in such a change process is guaranteed in the RENISYS method by performing the proper calculations on the set of applicable composition norms (see [5]). These norm calculations say which users may, must, or may not be involved in these knowledge definition change processes. For instance, there may be a norm that says that all system administrators must be involved in the creation

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

of new access-definitions. A change proposal can be in any part of the socio-technical system.

Example Instead of using FTP to upload files in the paper submission process, the project coordinator proposes to use BSCW. This means that the support for workflow mapping WM1 needs to be changed.

3. Formulate Matching Criteria Many different kind of functionality matches are conceivable. *Matching criteria* (or constraints) need to be specified on which the match is to be performed. Such criteria are expressed in terms of the elements of the meta-model. For instance, one criterion could be that when upgrading a tool by installing a new version (i.e., changing its enabled functionality), all existing workflow mappings that the old version supports must still be supported after the change. Once formulated, each criterion needs to be expressed in one or more *matching criteria graphs*. These graphs are the CG-queries necessary for retrieving the knowledge definitions that satisfy the matching criteria.

Example The change process concerns the replacing of tool instances in support-definitions (i.e., definitions that say which users use what tool instances to enable a particular workflow mapping). The matching criteria are (1) all tool instances of FTP in support-definitions of WM1 need to be replaced by tool instances of BSCW. Before this can be done, however, (2) all users that are part of the support definitions selected in (1) need to have an access-relation to at least one instance of the BSCW-tool. In this way, their requirements continue to be enabled. The accompanying matching criteria graphs are:

```
(1) [State: [Support] -
         (Inst) -> [FTP]
         (Obj) -> [Workflow_Mapping: #WM1]]
(2) [State: [Access] -
         (Poss) <- [User:#x]
         (Obj) -> [BSCW]]
```

4. Calculate the Match Using the functionality specifications of step 1 and the matching criteria graphs of step 3, the actual match is calculated. In general, such a match can be calculated by projecting the matching criteria graphs on the knowledge base of functionality specification graphs.

Example Matching criteria graph (1) is first projected on the specification knowledge base. Using the specialisations function of the Peirce workbench, the following result is returned:

```
> (Specialisations) -> [[State: [Support] -
   -> (Inst) -> [FTP]
   -> (Obj) -> [Workflow_Mapping:#WM1]]]?
[State: [User: #Mary]->(Poss)->[Support: #302]-
    (Inst)->[FTP: #EJCL]
    (Obj)->[Workflow_Mapping: #WM1],].
[State: [User: #John]->(Poss)->[Support: #300]-
```

```
    (Inst)->[FTP: #EJCL]
    (Obj)->[Workflow_Mapping: #WM1],].
true
>
```

This means that two users, Mary and John, currently make use of an FTP server.

Next, the matching criteria graph (2) is projected in similar fashion on the knowledge base, with *x replaced by #Mary and #John, respectively. Only for Mary, a specialization is returned. This means that she already has access to the BSCW tool, but John not yet.

5. Interpret the Matching Results Based on the criteria of step 3, the matching results of step 4 can be interpreted in different ways. Different *courses of action* can be taken to deal with functionality mismatches. For example, if one criterion says that no users should have access to a particular type of tool, then nothing needs to be done if no results are returned in step 4, whereas otherwise one or more functionality specifications may need to be redefined.

Example Since for John no access-relation has been returned, there first must be a specification process that gives him access to the BSCW-tool. To do so, an e-mail could automatically be sent to the system administrator. After access has been granted by means of an access-definition, the now superfluous definitions that described the FTP-support for the upload-file process can be removed.

4.2 Discussion

The functionality matching meta-model was based on the semantics introduced in the RENISYS specification method, which was explained in detail in [5]. In the literature, such a meta-model plus approach for supporting virtual communities in the specification of their network information systems was lacking at the time. Extensions are needed in various directions to realize a practical methodology. For example, we now assume that semantic mismatches between the required and the enabled functionality specification have already been resolved. In reality, much middleware consists of functionality components, such as information services, that are much more complex than the heavily simplified information and communication processes described in this paper. Furthermore, for implementation purposes links to low-level technical functionality specifications need to be established.

Another required extension is to expand the functionality matching metamodel with roles. In the current approach, users (e.g. John and Mary) are directly coupled to information tools. However, roles are an important construct for functionality specification to become more efficient. Roles can be loosely defined as collection of information and communication processes that can be performed by an *actor*. An *actor role*, such as an editor, can be played by various users at the same time. In our view, this concept enhances the matching process by limiting the necessity to determine for each individual user its workflow mappings and tool assignments.

Another limitation of the current approach is that only a few dependencies between specifications have been modelled so far. For example, besides the basic assignment

dependencies, there are many others conceivable. One issue concerns the relations between client and server tools: installing a BSCW server also means that users need to have a BSCW client (i.e. Web browser). This dependency has not been modelled yet.

We do not claim that from a theoretical perspective, this RENISYS-based approach is the only or even the best possible one. However, we do claim that the issues raised and elements of the functionality matching approach introduced are relevant in all matching approaches.

Once implemented and sufficiently extended, we also think that the functionality matching approach could become a true application, in the sense of [4]. Such an application should aid in the solution of actual problems, and be more than just a tool. Generic CG tools already exist and can be used to provide the basic functionality of the application. An important application area of our approach could be in developing testbeds [2], such as envisioned in the PORT project [1], in which many members of the CG-community are involved and which aims to develop a testbed methodology: "...The testbed methodology in a collaboratory research program provides a *virtual observational context* in which to study the needs of collaborators (in remote interaction with instruments, colleagues, and data) and to develop technology in response to those needs, for testing in that context. In testbeds, those collaborating must be able to monitor themselves in the process of examining how a proposed technology might augment their work."[5]. We feel that our approach, including its meta-model, could help to provide such a virtual observational context. One tool we are currently experimenting with is WebKB[6]. This tool seems to be well suited to construct such testbed applications, since it combines relatively advanced graph operations with a user-friendly, web-based interface. In this way, for example pulldown-lists can be easily generated with options for users to choose from, i.e. the list values are derived from graph operations on the knowledge base.

5 Conclusions

In this paper, we introduced a concrete functionality matching approach that aims to support virtual professional communities in order to achieve a more adequate evolution of their network information systems. The approach is based on a meta-model containing a detailed high-level, socio-technical semantics of the relations between requirements in the form of information and communication processes, users, and information tools. The approach was illustrated by a real-world case: an electronic law journal.

The functionality matching approach proposed here bridges two theoretical worlds. It is on the one hand related to work on component interface matching, which currently dominates middleware research. A major drawback of existing approaches is that they are defined at a very low level and do not contain any semantics of the evolution of the socio-technical system of virtual professional communities. On the other hand, our approach makes use of the power of conceptual graph theory, notably the availability of graph generalization hierarchies for efficient specification representation and easy

[5] Proposal for Workshop on the Semantic Web for ICCS 2001, PORT-mailing list, 24 December 2000

[6] http://www.webkb.org

calculation of graph matches by means of basic projection operations. Of course, the proposed approach is only a very simple one. The most important contribution currently is that the approach (1) makes explicit use of a functionality matching meta-model to describe high level socio-technical semantics; it recognizes that different communities (2) may apply different matching criteria, so that they can define their own, customized constraints on the evolution of their socio-technical system and (3) interpret the results in their own way by taking potentially different courses of action in case of violation of matching constraints. This tailored approach to defining the implementation of network information systems does justice to the unique and volatile nature of many virtual communities.

References

1. M.A. Keeler, C. Kloesel and L. Searle. PORT: A Testbed Paradigm for Knowledge Processing in the Humanities, *in: Lecture Notes in Artificial Intelligence*, vol. 1257, Springer-Verlag, pp. 100-113, 1997.
2. J. Lederberg and K. Uncapher. Towards a National Collaboratory: [NSF] Report of an Invitational Workshop. *Rockefeller University, New York City, 13P15, March 1989*.
3. W.J. van den Heuvel, M.P. Papazoglou and M. Jeusfeld. "Connecting Business Objects to Legacy Systems", Proceedings of the CAiSE Conference, Springer, 1999.
4. M. Chein, D. Genest. CG Applications: Where are We 7 Years after the First ICCS? In *Proceedings of the Eighth International Conference on Conceptual Structures, ICCS2000, Darmstadt, Germany, August 14–18, 2000*, 2000, pages 127–139.
5. A. De Moor. Composition norm dynamics calculation with conceptual graphs. In *Proceedings of the Eighth International Conference on Conceptual Structures, ICCS2000, Darmstadt, Germany, August 14–18, 2000*, 2000, pages 525–539.
6. O. de Troyer, R. Meersman, and P. Verlinden. RIDL on the CRIS case: A workbench for NIAM. In T.W. Olle, A.A. Verrijn-Stuart, and L.. Bhabuta, editors, *Computerized Assistance During the Information Systems Life Cycle*, pages 375–459. Elsevier Science Publishers, B.V., 1988.
7. T.A. Finholt and G.M. Olson. From laboratories to collaboratories: A new organizational form for scientific collaboration. *Psychological Science*, 8(1):28–36, 1997.
8. B.R. Gaines, L.J. C. Lee, and M.L.G. Shaw. Modeling the human factors of scholarly communities supported through the Internet and the World Wide Web. *Journal of the American Society for Information Science*, 48(11):987–1003, 1997.
9. N.C. Goodwin. Functionality and usability. *Communications of the ACM*, 30(3):229–233, 1987.
10. E. Bertino. "Integration of Heterogenuous data repositories by using object-oriented views", ACM Transactions on Database Systems, Vol. 17(3):385-422, 1992.
11. S. Chen. Retrieval of Reusable Components in a Deductive, Object-Oriented Environment. PhD thesis, RWTH Aachen, Information Systems Institute, 1993.
12. Scott Henninger. An Evolutionary Approach to Constructing Effective Software Reuse Repositories", ACM Transactions on Software Engineering and Methodology", vol. 6, nr. 2, pp. 111–140, 1997.
13. L. Kalichenko. Workflow Reuse and Semantic Interoperability Issues, *Worklfow Management Systems and Interoperability,* A. Doğaç, L. Kalichenko, M.T. Őzsu and A. Seth, NATO ASI Series, Springer, 1998.

14. G. Spanoudakis and P. Constantopoulos. Similarity for Analogical Software Reuse: A Conceptual Modelling Approach, in: Proceedings of the 5th International Conference on Advanced Information Systems Engineering, (CAiSE '93), (eds) Rolland C., Cauvet C., LNCS 685, Springer -Verlag, 1993
15. Am.M. Zaremski and J.M. Wing. Specification Matching of Software Components. ACM Transactions on Software Engineering and Methodology, Vol. 6, No. 4, pp. 333-369, Oct. 1997.