

# Chapter 11

---

## Optimizing Social Software System Design

**Aldo de Moor**

*CommunitySense, Tilburg, The Netherlands*

11.1 Introduction .....	279
11.2 Social Software: from Tools to Systems .....	280
11.3 A Conceptual Model of Functionality Matching in Collaborative Communities .....	287
11.4 The Functionality Matching Process .....	291
11.5 Discussion and Conclusion .....	297
11.6 Student Projects .....	298

---

### 11.1 Introduction

Collaborative communities are important catalysts of research, economic, and social processes. In these communities, many stakeholders collaborate on joint goals, although often having partially conflicting interests. Examples are research communities, knowledge management teams, innovation platforms, and environmental campaign networks. These communities make use of an ever widening range of social software tools, including many types of discussion fora, blogs, content management systems and advanced knowledge analysis and processing tools. These tools increasingly come with Web 2.0 functionalities like tagging and reputation management systems.

Collaborative communities are complex and rapidly evolving socio-technical systems. The design of these systems includes the communal specification of communication and information requirements, as well as the selection, configuration, and linking of the software tools that best satisfy these requirements. Supporting the effective and efficient community-driven design of such complex and dynamic systems is not trivial.

To represent and reason about the system design specifications we use conceptual graph theory. We do so because the knowledge representation language of choice must be rich enough to allow the efficient expression of complex definitions. Also, since design specifications derive from complex real world domains and community members themselves are actively involved in specification processes, a close mapping of knowledge definitions to natural language

expressions and vice versa is useful. Finally, the representation language must be sufficiently formal and constrained for powerful knowledge operations to be constructed. Conceptual graph theory has all of these properties.

In this chapter, we explore how conceptual graphs can be used to:

1. model the core elements of such socio-technical systems and their design processes.
2. specify communication and information requirements and match these with social software functionalities.

We illustrate these design processes with examples from a realistic scenario and end the paper with a number of suggestions for student projects to extend the ideas proposed in this chapter.

Sect. 11.2 summarizes our view of social software, which we describe as an evolving socio-technical system consisting of a *tool system* embedded in a *usage context*. Using these notions, in Sect. 11.3 we present a conceptual model of functionality matching in collaborative communities. Sect. 11.4 shows how the functionality matching process can be supported using these conceptual definitions. Sect. 11.5 contains a discussion and conclusion. Sect. 11.6 lists a set of student projects expanding the material introduced.

---

## 11.2 Social Software: from Tools to Systems

Collaborative communities, in which people work together to accomplish common goals, make use of a wide variety of social software tools to support their information processing, communication, and coordination needs. A typical configuration would include a content management system, some mailing lists, a bulletin board, and increasingly more sophisticated Web 2.0 tools like LinkedIn<sup>1</sup> for maintaining professional contact networks, Digg<sup>2</sup> for filtering relevant Web documents, and so on. In de Moor [2007], we presented a conceptual model of a tool system and its usage context as a way to describe such socio-technical systems. Here we briefly summarize the main elements.

We define a *tool system* as the set of integrated and customized information and communication tools tailored to the specific information, communication, and coordination requirements of a collaborative community. There are numerous, partially overlapping implementations of such functionalities. In addition, many of these tools are built on top of an emerging cyberinfrastructure of organizational practices, technical infrastructure, and social norms

---

<sup>1</sup><http://www.linkedin.com>

<sup>2</sup><http://www.digg.org>

Edwards et al. [2007]. Furthermore, each community has its own, unique way of using these functionalities. Finally, the requirements and technologies used are in constant flux. In all, this makes it extremely complicated to come up with standardized prescriptions for the best tool system for a particular community at a particular moment in time. New forms of analysis, roles in software development, and the meaning of use and maintenance need to evolve Sawyer [2001], which applies even more to the case of community information systems development.

To design useful information systems by selecting, linking, and configuring the right components, available tool system functionalities in the form of modules and services need to be evaluated in their context of use by the communities of use themselves. Effective use is key here, which can be defined as the capacity and opportunity to successfully integrate ICTs into the accomplishment of self or collaboratively identified goals Gurstein [2003]. Evaluating the usefulness of a functionality can be defined as the evaluation of the extent to which users can translate their intentions into effective actions to access the functionality Gaines et al. [1997]. Such evaluation should ultimately contribute the purpose of the community of use Preece [2000]. Such an approach requires the continuous comparison of tool functionalities with usage context requirements, for which this chapter provides some building blocks.

We first present a hypothetical scenario of a knowledge-driven topic community on climate change, which will be used to illustrate the ideas introduced in this chapter. We then present a conceptual model of both the tool system and the usage context, together making up the socio-technical system of a collaborative community.

### 11.2.1 Scenario: Building a Knowledge-Driven Topic Community on Climate Change

Scientific consensus is growing rapidly that climate change is indeed happening and will have a serious societal impact. However, the causes and effects of this phenomenon are still ill-understood and need urgent and concerted analysis by many stakeholders from all over the world. Let us assume that the United Nations Environment Programme (UNEP) has been commissioned to - as quickly as possible - create a range of task groups to examine scientific consensus on the causes and effects of climate change and to make appropriate policy recommendations. The task groups are to be broad in scope, including scientific experts as well as opinion leaders from business, governments, NGOs and the general public. Over time, these ad hoc task groups should grow into well-established, collaborative topic communities, able to rapidly evaluate highly complex and contradictory results and to efficiently inform policy makers and other stakeholders all over the world.

Jane, a senior UNEP official, is in charge of growing these topic communities. Having selected an initial topic community charged with examining

causes and effects of ice cap melting<sup>3</sup>, she now faces the problem of defining the appropriate tool system to support their collaboration.

The first task of the community is to take stock of the relevant climate change entries in Wikipedia<sup>4</sup>, as this is a well-established repository of basic knowledge on the main issues, with contributions made by numerous (self-appointed) experts and stakeholders worldwide. As these authors represent a wide range of, often contradictory, opinions, and since Wikipedia has such a high visibility, the knowledge represented in the Wikipedia entries forms a good starting point for more in-depth analysis and dissemination by the topic community.

The Wikipedia network of hyperlinked entries forms an informal *concept network*. However, many of the nodes contain irrelevant or wrong information from the point of view of the topic community. Furthermore, other interesting links between items within Wikipedia and with information resources on the Internet at large could be conceived. Finally, this resource only provides the raw information for the topic community's goal: advising on policy and decision making. What is needed is a sensemaking, filtering, and knowledge recombination and extension *discourse* between various topic community members. We call these discussion foci *discourse topics*, which in this case include causes and effects of and policy recommendations for addressing ice cap melting. The discourse topics thus are orthogonal to the knowledge entries, one topic potentially linking to many entries and vice versa. Several domain and discussion roles are defined as well. Representatives of relevant societal stakeholders should be included in the topic community. Topics are to be discussed by discussants, facilitators should keep the complex discourse process on track, and summarizers should formulate consensus positions for dissemination to the general public. To support this discourse process, Jane opts for the Compendium tool<sup>5</sup>. This is a visual discourse-supporting hypermedia tool for sensemaking that provides a variety of mechanisms for analysts to tag and connect media assets in relation to issues, ideas and arguments. Fig. 11.1 outlines the socio-technical system of the topic community.

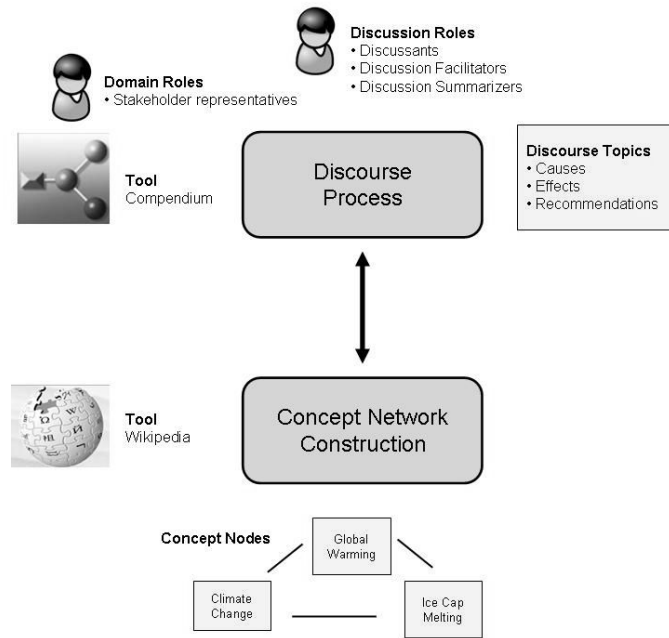
Having decided on these principal tools, Jane now faces the design problem of determining how the workflows should look that these tools are to support, which functionality components of the tools are to support what workflow steps, and how these functionality components should be integrated or at least aligned.

---

<sup>3</sup>In de Moor and Anjewierden [2007], we presented a socio-technical approach for the initial stage of this process: the selection of candidate members for these topic communities by using the tOKo tool for text analysis to mine the blogosphere.

<sup>4</sup>e.g. [http://en.wikipedia.org/wiki/Climate\\_change](http://en.wikipedia.org/wiki/Climate_change)

<sup>5</sup><http://www.CompendiumInstitute.org>



**FIGURE 11.1:** Outline of the Topic Community Socio-Technical System

### 11.2.2 The Tool System

A functionality is a set of functions and their specified properties that satisfy stated or implied needs<sup>6</sup>. Functionalities come in different forms of aggregation, from simple word processing functions to complete modules and applications supporting particular workflows or business processes. When evaluating functionality, the right level of granularity should be chosen.

In Christiaens and de Moor [2006], we introduced a basic model of a *tool system*, consisting of components of different levels of granularity. At the lowest level of granularity, we distinguish *systems* of tools or services. The next level consists of the *tools* or services themselves. Then come the *modules* comprising the tools or services. Finally, we distinguish the particular *functions* grouped in a module.

Functionality components of different levels of granularity can be linked in many ways. The *interface* between two functionality components A and B is described by the externally visible representation of the knowledge structures and processes used in the interoperating components A and B. It is important only to model such high-level conceptual structures when designing the tool

<sup>6</sup>Definition by the Software Engineering Institute Open Systems Glossary: <http://www.sei.cmu.edu/opensystems/glossary.html>

system, to prevent unnecessary modeling and interpretation efforts in this initial stage. Only when the total system architecture has been agreed upon, should further modeling efforts be made<sup>7</sup>.

In Christiaens and de Moor [2006], we focused on the communication aspects of interoperating functionality components. We abstracted from the internal *information processes*. These are operations on information objects within a functionality component, such as the creation, modification, or deletion of an object, such as editing an HTML page. For tool selection, such processes may be important, however, so we include them in our conceptual model. Again, note that we are only interested in those information processes visible to a user, so we abstract from all internal computational processes.

*Example:*

Wikipedia's main unit of analysis is the *page*. Each page contains a textual description of the topic, and can contain *links* to other pages as well as *categories* indexing the content. Two functionality modules of particular interest are the *Search Pages* and *Edit Page*. Functions within the Edit Page module include *Edit Text*, *Edit Link*, and *Edit Category*.

Compendium has many different functionality modules. The one of particular interest to our purpose concerns *Create Node*, with functions like *Create Question*, *Create Answer*, *Create Argument*, *Create Decision*, *Create Reference*, etc.

### 11.2.3 The Usage Context

In de Moor [2007], we distinguished three important categories of ontological elements making up the usage context: goals, actors, and various domain factors. We summarize these elements here.

#### 11.2.3.1 Goals

Goals or objectives are crucial in the pragmatic view. Everything starts with goals. Goals give a sense of purpose, drive people and processes, and can be used as evaluation criteria.

We distinguish two types of goals. First, there are *activities*, such as "writing an advisory report" or "making a climate change argumentation analysis".

---

<sup>7</sup>Note that in this chapter, we first discuss the tool system, then the usage context. Strictly speaking, one should first analyze the requirements determined by the context, and only then determine which functionalities match best. However, we express requirements in terms of mappings to functionalities. Furthermore, our approach is an evolutionary one, which means that legacy functionalities often pre-determine the degrees of freedom in functionality selection. Socio-technical systems analysis is a continuous and delicate balancing act between requirements and functionalities, which in practice is much more a parallel than a serial process.

Such activities in fact are operationalized goals: processes with a concrete deliverable as an outcome, often in the form of an information object such as a report or a message. However, many goals are abstract and cut right across processes and structures. Examples of such goals are non-functional requirements and quality criteria, like "efficiency", "participation rate", and so on. We call such abstract goals *aspects*.

Although activities can be viewed as workflows, we abstract from their design and implementation details, such as concurrency and sequence. Although definitely important in the final construction of the information system, for the purpose of functionality component selection they add unnecessary complexity. This initial stage, the focus of this paper, concerns itself with the selection of *potential* functionalities, not their actual configurations for workflow support.

*Example:*

In the scenario, the following *activities* are distinguished:

1. Select relevant concepts and their entries in the knowledge system
2. For each discourse topic, conduct a discussion among relevant stakeholders, resulting in a consensus position.
3. Disseminate the consensus position to the general public.

One quality *aspect* which should be ensured by the tool system is *legitimacy*. This is defined as that all actors should be involved in those and only those activities that are relevant to their discussion or domain roles.

### 11.2.3.2 Actors

Many stakeholders are involved in tool system design and use. For example, in the domain of courseware evaluation, students, teachers, the computer center, etc. all have their specific, often partially incompatible interests, needs, and goals. Students prefer easy-to-use functionalities, whereas main concerns of the computer center include security and reusability de Moor [2007]. In order to ensure that all requirements are captured, contrasted, and balanced, it is not sufficient to examine "The User". Instead, an inventory carefully needs to be made of the *actor roles* that the various stakeholders play. Making roles explicit is gaining prominence in, for instance, the Role Based Access Control paradigm<sup>8</sup> as a way to systematically determine actor responsibilities in workflows and access to functionalities and information resources. Most role classifications are quite abstract and technology-focused (administrator, facilitator, member, etc.). However, many other typologies are possible, often

<sup>8</sup><http://csrc.nist.gov/rbac/>

quite specific to a particular domain. Roles could, for instance, be based on workflow process (author, editor, reviewer, ...), on organizational structure (secretary, manager, ...), or on the main stakeholders in a particular domain (UNEP, Corporation, NGO, ...). Precisely defining the responsibilities, permissions, and prohibitions attached to these roles helps in creating better design patterns of information systems. Besides having their normal operational status, such actor roles should also be explicitly involved in the design and optimization of the socio-technical system-in-use.

*Example:*

As the goal of the topic community is to provide policy advice, it is imperative that voices across society have their say. The *domain role* of representative is therefore introduced. Further analysis would need to specify which stakeholders need representation: science, business, government, NGOs, the general public, and so on. To ensure the efficiency and effectiveness of the required discussion processes, the following *discussion roles* are necessary: discussant, discussion facilitator, and discussion summarizer.

### 11.2.3.3 Domain

A third layer of the model is the domain in which the collaborative community using the tool system (inter)operates. The domain is an important, but still ill-understood element in tool system design and evaluation. Aspects that influence design processes and functionalities of the tool system include issues of structure and size, for example is it a distributed network or centralized organization, a large or a small organization? What setting is the tool being used in: an academic, a corporate, a governmental, or a non-governmental setting? What is the financial situation: are there resources for acquisition or customization of software, or is off-the-shelf, open source software the only option? Are there political alliances and commitments that force or preclude the use of certain software? In other words, the domain determines how many degrees of freedom there are in design process.

Such domain issues are not likely to be translated into goals like activities and aspects directly, but they do help to define the scope of the project and the general affordances and constraints in which the tool system needs to (inter)operate.

*Example:*

The topic community concerns a distributed, medium sized network, in which the mode of working in general will be asynchronous and dispersed due to the global nature of the community and the almost impossible matching of agendas. It concerns an intergovernmental setting, with much attention to procedure, legitimacy, participation, and accountability, features that the

tool system should reflect. As with many (inter)governmental bodies, there is some money for customization and process facilitation, but no budget to buy expensive, proprietary tools. Also, the UN has a strong interest in promoting free software and prefers to lead by example. Well-documented, reliable, and interoperable open source tools are therefore strongly preferred.

---

### 11.3 A Conceptual Model of Functionality Matching in Collaborative Communities

In this section, we formalize and extend the notions introduced so far. First, we present a concept type hierarchy of the socio-technical system. We then use this hierarchy to define some key functionality mappings using conceptual graphs. Then, we present a worked example of the functionality matching process based on these mappings.

#### 11.3.1 A Concept Type Hierarchy of the Socio-Technical System

Based on the scenario, the following concept type hierarchy can be created to model the socio-technical system of the topic community.

*Actors* either play discussion or domain roles. *Discussion roles* include *Discussants*, *Discussion Facilitators*, and *Discussion Summarizers*. *Domain roles* include *Representatives* of various stakeholders, which (as of yet) do not need to be formalized in more detail.

*Functionality Components* of different levels of granularity are distinguished. The *Topic Community Support System* is the focus of attention. The two main *Tools* used are *Compendium* and *Wikipedia*. Each tool contains one or more *Modules*, such as *Create Node*, *Edit Wiki*, or *Search Pages*. Each of these modules contains a number of *Functions*, which themselves are further grouped in categories like *Create a Text Element*, *Create Links* to and from text elements, or *Create Indices* of text elements.

*Goals* of the system are the *Activities* of *Selecting Relevant Concepts* and their entries in the knowledge system, *Conducting a Discussion* among relevant stakeholders, and *Disseminating the (consensus) Position* to the general public. One important Aspect is the *Legitimacy* of participation in the activities.

*Mappings* within and between elements of the tool system and usage context include *Enabled Functionalities*, *Functionality Requirements*, *Required Implementations*, and *Support-definitions*.

```
T >
  Actor >
    Disc_Role >
      Module >
        Create_Node
        Edit_Wiki
```

Discussant	Search_Pages
Disc_Facilitator	System >
Disc_Summarizer	Topic_Community_Support_Sys
Domain_Role >	Tool >
Representative	Compendium
Funcnt_Comp >	Wikipedia
Function >	Goal >
Create_Index >	Activity >
Create_Category	Conduct_Disc
Create_Tag	Disseminate_Pos
Create_Link >	Sel_Rel_Concepts
Create_Reference	Aspect >
Create_Text >	Legitimacy
Create_Argument	Mapping >
Create_Arg_Con	Enable
Create_Arg_Pro	Funcnt_Req
Create_Position >	Req_Impl
Create_Answer	Support
Create_Decision	
Create_Idea	
Create_Issue >	
Create_Question	
Create_Note	
Edit_Disc_Page	
Edit_Page	

### 11.3.2 Functionality Mappings

Functionality mappings define relations within and between various elements of the usage context and the tool system, and are used in the functionality matching process. In de Moor and van den Heuvel [2001], we presented a functionality matching meta-model and process for virtual communities, using conceptual graphs to assess the match between required and enabled functionalities. Here, we present an adapted version of that approach<sup>9</sup>.

We first propose a set of effective tool functionality axioms, which help establish the link between tool system and usage context.

#### Effective Tool Functionality Axioms

- A functionality component can enable one or more functions.  
**Example:** the Compendium *Create Node*-module allows a user to create an issue, a position, an argument, etc.
- Different functionality components may have partially *overlapping* functionality, i.e. each enabling the same functions, while also each enabling different functions at the same time.  
**Example:** Both Compendium and Wikipedia allow for links between knowledge

<sup>9</sup>The most important changes are: including roles instead of specific users in the definitions, simplifying the workflow mappings by leaving out the interaction layer, using functions instead of information/communication processes as the atomic unit of functionality, and not including access-relations. The reader is referred to the original paper for more detail.

nodes to be created. However, only Compendium also allows for visually mapping the structure of a debate, while only Wikipedia provides users with the capabilities to jointly efficiently edit and review complex wiki knowledge nodes.

- All community members involved in a functionality requirement must have at least one *enabling functionality component* at their disposal.

**Example:** a community member may need to be able to create links between knowledge nodes. Both Compendium and Wikipedia provide their users with this functionality.

### Enabled Functionality

Any function enabled by some functionality component for a particular actor role is called an *enabled functionality*. Such a mapping is represented as a definition which conforms to a specialization of the following definition of the enabled functionality-mapping<sup>10</sup>:

$$\begin{aligned} &[\text{Enable} : *x] \rightarrow (\text{Def}) \rightarrow [\text{Mapping} : ?x] - \\ &(\text{Inst}) \rightarrow [\text{Tool}] \rightarrow (\text{Part}) \rightarrow [\text{Module}] \rightarrow (\text{Obj}) \rightarrow [\text{Function}] \\ &(\text{Agnt}) \rightarrow [\text{Actor}]. \end{aligned}$$

*Example:*

The following definition says that creating an argument is enabled by the Compendium *Create Node*-module for any user of the component:

$$\begin{aligned} &[\text{Enable} : \#114] - \\ &(\text{Inst}) \rightarrow [\text{Compendium}] \rightarrow (\text{Part}) \rightarrow [\text{Create\_Node}] - \\ &(\text{Obj}) \rightarrow [\text{Create\_Argument}] \\ &(\text{Agnt}) \rightarrow [\text{Actor}]. \end{aligned}$$

### Required Functionality

Functionality requirements are defined by functions in their usage context, which we define as the activity for which a function is used and the actor role involved in that activity:

$$\begin{aligned} &[\text{Funct\_Req} : *x] \rightarrow (\text{Def}) \rightarrow [\text{Mapping} : ?x] - \\ &(\text{Obj}) \rightarrow [\text{Activity}] \rightarrow (\text{Agnt}) \rightarrow [\text{Actor}] \\ &(\text{Inst}) \rightarrow [\text{Function}]. \end{aligned}$$

<sup>10</sup>First a type definition of each concept is given. The (Def) relation connects the defined type on the left with the genus (supertype) on the right, while the rest of the definition comprises the differentia, the subgraph that specialises the genus to the defined type. The knowledge definitions in the examples are specializations of defined type plus differentia

*Example:*

The following definition says that for conducting a discussion a discussion summarizer needs to be able to create a position.

```
[Funct_Req : #156]–
  (Obj) → [Conduct_Disc] → (Agnt) → [Disc_Summarizer]
  (Inst) → [Create_Position].
```

### Assigned Functionality

For each functionality requirement, at least one enabling functionality component needs to be assigned. The actual selection of the functionality components that are to enable functionality requirements is not automated in our approach. The reason is that there may be many domain and non-functional requirements constraining the choice of implementation, which require human interpretation. Two types of assigned functionality include support-mappings and required implementation-mappings.

### The Support-Mapping

A *support*-mapping represents a function supporting a functionality requirement.

```
[Support : *x] → (Def) → [Mapping :?x]–
  (Obj) → [Funct_Req]
  (Inst) → [Tool] → (Part) → [Module] → (Obj) → [Function].
```

*Example:*

The following support-relation says that for a discussion summarizer to be able to create a position (Functionality Requirement #156) she can use the *Create Decision*-function of the *Create Node*-module of *Compendium*.

```
[Support : #212]–
  (Obj) → [Funct_Req : #156]
  (Inst) → [Compendium] → (Part) → [Create_Node]–
  (Obj) → [Create_Decision].
```

### The Required Implementation-Mapping

Legacy systems and the need to standardize implementations for maintenance efficiency often lead to the demand for certain functionality requirements to be supported by a specific tool. Such a *required implementation* puts an additional constraint on possible support-relations.

$$\begin{aligned}
 &[\text{Req\_Impl} : *x] \rightarrow (\text{Def}) \rightarrow [\text{Mapping} : ?x] - \\
 &(\text{Obj}) \rightarrow [\text{Funct\_Req}] \rightarrow (\text{Obj}) \rightarrow [\text{Activity}] \\
 &(\text{Inst}) \rightarrow [\text{Tool}].
 \end{aligned}$$

Note that for readability we repeat the activity-concept which is already part of the functionality requirement.

*Example:*

The following required implementation-mapping says that for any functionality requirement concerning the conducting of discussions, the Compendium tool needs to be used.

$$\begin{aligned}
 &[\text{Req\_Impl} : \#186] - \\
 &(\text{Obj}) \rightarrow [\text{Funct\_Req}] \rightarrow (\text{Obj}) \rightarrow [\text{Cond\_Disc}] \\
 &(\text{Inst}) \rightarrow [\text{Compendium}].
 \end{aligned}$$


---

## 11.4 The Functionality Matching Process

Using the functionality mappings, we now examine the steps of the *process* in which required and enabled functionalities can be matched in practice.

Already with these very basic functionality mappings, essential functionality matching operations can be performed. The real power of conceptual graphs, however, is that on top of these definitions, a wide range of (meta)-constraints can be defined, at various (meta)-levels of analysis Mineau et al. [2000]. Such constraints are especially important when operationalizing abstract quality aspects, such as efficiency, extensibility, security, and legitimacy<sup>11</sup>. Bootstrapping the complex definition of such requirements with conceptual graphs could significantly help optimize the design process.

The matching process consists of three stages: (1) creating a knowledge base of socio-technical system specifications, (2) proposing some change to the specifications, and (3) performing the match (which includes formulating a set of functionality matching criteria, calculating the match, and interpreting the results).

---

<sup>11</sup>We gave legitimacy as an example of a goal-aspect. We do not work out its operationalization here, but have defined it as a student project at the end of this chapter.

### 11.4.1 Define System Specifications

Collaborative communities are continuously evolving socio-technical systems. At  $t=0$ , before a change is proposed, we assume that the current system specifications are properly matched, i.e. that all required functionalities are enabled, and that all required implementation-constraints are satisfied.

*Example:*

Assume the concept type hierarchy and definitions given above. Furthermore, the following graphs are given.

#### Enabled functionality:

This definition captures (part of) the functionality provided by the Compendium *Create Node*-module:

```
[Enable : #115]–
  (Inst) → [Compendium] → (Part) → [Create_Node]–
    (Obj) → [Create_Answer]
    (Obj) → [Create_Idea]
    (Obj) → [Create_Argument]
    (Obj) → [Create_Arg_Con]
    (Obj) → [Create_Arg_Pro]
    (Obj) → [Create_Decision]
    (Obj) → [Create_Note]
    (Obj) → [Create_Question]
    (Obj) → [Create_Reference]
  (Agt) → [Actor].
```

The next definition captures (part of) the functionality provided by the Wikipedia *Edit Text* and *Search Pages*-modules<sup>12</sup>.

```
[Enable : #116]–
  (Inst) → [Wikipedia]–
    (Part) → [Edit_Wiki]–
      (Obj) → [Create_Category]
      (Obj) → [Create_Link]
      (Obj) → [Edit_Disc_Page]
      (Obj) → [Edit_Page]
    (Part) → [Search_Pages].
  (Agt) → [Actor].
```

<sup>12</sup>Note that the functions of the *Search Pages*-module has not been defined. However, including this module-concept in the definition shows that there is at least *some* searching-functionality, which could be investigated and explicated further if required.

**Assigned functionality:**

Support-definition #212 stated that Compendium's Create Position-function is to be used to support the Discussion Summarizer in her participating in the discussion. Required Implementation-definition #186 said that every functionality requirement having to do with conducting discussions has to be supported by Compendium. In addition, the following definition expresses that disseminating the final position to the general public debate should be supported by Wikipedia:

```
[Req_Impl : #187]-
  (Obj) → [Funct_Req] → (Obj) → [Disseminate_Pos]
  (Inst) → [Wikipedia].
```

**11.4.2 Propose Specification Changes**

At  $t=1$ , one or more specification changes are 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 so far. In collaborative communities, it is essential *who* has the legitimate authority to make such changes: in de Moor and Weigand [2007], we show how such legitimacy can be ensured by calculating these authorizations using the set of *applicable composition norms*. The calculation says which community members may, must, or may not be involved in the specification of particular parts of their socio-technical system. In this chapter, we unfortunately do not have the space to say more about this highly relevant, but complex topic.

We illustrate the processing of a newly specified (legitimate) functionality requirement to show how conceptual graphs can help optimize the design process.

*Example*

A discussion summarizer, besides having to be able to create a position during the conduct of the discussion also must be able to disseminate the position in the right form to the general public. As it is not clear yet in what form this dissemination best take place, all that can be said for now is that it should be some form of text creation, to be specified in more detail later, represented by this new functionality requirement #158:

```
[Funct_Req : #158]-
  (Obj) → [Disseminate_Pos] → (Agnt) → [Disc_Summarizer]
  (Inst) → [Create_Text].
```

The question is how to assign the appropriate enabled functionality? To answer this question, we need to match required and enabled functionalities.

### 11.4.3 Perform the Functionality Matching

Functionality matching can help to correctly process specification changes, such as new functionality requirements, ensuring the legitimacy, consistency and completeness of the definitions of the evolving socio-technical system.

Many different forms of functionality matching are conceivable. *Matching criteria graphs* (or constraints) need to be specified on which the matching steps are to be performed. The matching criteria should at least partially be expressed in terms of the functionality mappings. These graphs are the CG queries necessary for retrieving the knowledge definitions that satisfy the matching criteria. Often, a functionality matching process is complex, consisting of a sequence of matching graphs to be projected, joined, etc. The outcome of this process will be an answer to the question whether or not a specification change does have adverse effects on the socio-technical system, or provide suggestions for new specification changes. For example, if a new functionality requirement is introduced, there may not yet be the right enabling functionality. In that case, either the functionality requirement needs to be modified, a new functionality component needs to be installed, or at least a procedure needs to be in place with instructions for how to deal with the functionality problem.

In our example, the initial matching criteria graphs are the actor-concept (the concept in the graph that is the specialization of [Actor]) and the function-concept (the concept in the graph that is the specialization of [Function]) of each enabled functionality  $ef$  in the total set of enabled functionalities  $EF$ . To enable a particular functionality requirement  $fr$ , the function-concept of at least one  $ef_i \in EF$  should be a subtype of its counterpart in the functionality requirement  $fr$ , as it should be some implementation of the required function-concept. However, the actor-concept of that  $ef_i$  should also be a *supertype* of the actor-concept of  $fr$ , as it must at least be enabled for the required role.

#### 11.4.3.1 Functionality Matching Steps

For a particular functionality requirement  $fr$ , the functionality matching steps are as follows:

- Determine the set of potentially enabling functionalities  $EF_{pot}$  of  $fr$ . To do so, for each enabled functionality  $ef_i \in EF$ :
  - If the function-concept of  $fr$  projects into the function-concept of  $ef_i$  and the actor-concept of  $ef_i$  projects into the actor-concept of  $fr$ , then  $ef_i$  is in  $EF_{pot}$ .

- Determine the set of relevant required implementation-mappings  $RI$ , which are those required implementation-mappings of which the activity-concept projects into the activity-concept of  $fr$ .
- Determine the set of acceptable enabling functionalities  $EF_{acc}$ , which equals  $EF_{pot}$  minus those enabled functionalities  $ef_i \in EF_{pot}$  where none of the  $ri \in RI$  has a tool-concept that projects into  $ef_i$ . In other words, if a particular activity (as defined in  $fr$ ) is to be supported by one or more specific tools, but the potentially enabling functionality does not contain a specialization of any of those, that functionality is not acceptable.
- Create the support-mapping that defines which function enables  $fr$  by selecting one or more acceptable enabling functionalities from  $EF_{acc}$ .

### Example

We now conduct the functionality matches, using the CG projection operation. One tool that supports these operations is Prolog+CG, part of the Amine platform<sup>13</sup>.

We need to define a support-mapping for functionality requirement  $fr$  #158. The function-concept of  $fr$  is `[Create_Text]`, its actor-concept is `[Disc_Summarizer]`.

Projecting `[Create_Text]` into all  $ef \in EF$ , Prolog+CG returns the following enabled functionality-(sub)graphs<sup>14</sup>:

```
?- cg(_,g1),subsume([Enable]-inst->[Tool]-part->[Module]
    -obj->[Create_Text],g1,g2).

{_=FREE, g1=[Enable : nr114] -
  -inst->[Compendium]-part->[Create_Node]-obj->[Create_Argument],
  -agnt->[Actor],
  g2=[Enable : nr114]-inst->[Compendium]-part->[Create_Node]
  -obj->[Create_Argument]}

{_=FREE, g1=[Enable : nr115a]
  -inst->[Compendium]-part->[Create_Node] -obj->[Create_Answer],
  -agnt->[Actor];,
```

<sup>13</sup>Prolog+CG currently does not allow for non-functional relations to be processed. This means that graphs with more than one instance of the same relation emerging from a concept node, are not processed correctly. In a future version of Amine, the the tool environment embedding the newest versions of Prolog+CG, such non-functional relations will be interpreted correctly Kabba [2007]. For now, we need to flatten graphs like enabled functionalities #115 and #116 into their separate subgraphs with only a single (Part)-relation per subgraph.

<sup>14</sup>`cg(_,g1)` retrieves a graph from the knowledge base. The `subsume(g0, g1, g2)` operation then checks that `g0` subsumes `g1` and returns in `g2` the image of `g0` in `g1` (the subgraph of `g1` that is isomorph to `g0`)

```
g2=[Enable : nr115a]-inst->[Compendium]-part->[Create_Node]
  -obj->[Create_Answer]}
```

[...]

```
{_ =FREE, g1=[Enable : nr116d]
  -inst->[Wikipedia]-part->[Edit_Wiki] -obj->[Edit_Page],
  -agt->[Actor];,
  g2=[Enable : nr116d]-inst->[Wikipedia]-part->[Edit_Wiki]
  -obj->[Edit_Page]}
```

Enabled functionality-subgraphs #115i, #116a, and #116b are not returned, as their enabled functions (resp. `Create_Reference`, `Create_Category` and `Create_Link`) are not specializations of the `Create_Text`-concept of the functionality requirement  $fr$  to be supported. Since the actor-concept of all  $ef$  is `[Actor]` (the most generic role), all projections obtained just now are in  $EF_{pot}$ .

The activity-concept of  $fr$  is `[Disseminate_Pos]`. Required implementation #187 is the only one of which the activity-concept (also `[Disseminate_Pos]`) projects into the activity-concept of  $fr$ , and is therefore the only element in the set of relevant required implementation-mappings  $RI$ .

To calculate the set of acceptable enabling functionalities,  $EF_{acc}$ , we need to find those  $ef_i \in EF_{pot}$  where the tool-concept is a specialization of the tool-concept `[Wikipedia]` of required implementation #187:

```
?-cg(_,g1),subsume([Enable]-inst->[Tool]-part->[Module]-obj->
  [Create_Text],g1,g2),subsume([Wikipedia]-part->
  [Module]-obj->[Function],g2,g3).
```

```
{_ =FREE, g1=[Enable : nr116c]-inst->[Wikipedia]-part->[Edit_Wiki] -
  -obj->[Edit_Disc_Page],
  -agt->[Actor];, g2=[Enable : nr116c]-inst->[Wikipedia]-part->
  [Edit_Wiki]-obj->[Edit_Disc_Page], g3=[Wikipedia]-part->
  [Edit_Wiki]-obj->[Edit_Disc_Page]}
```

```
{_ =FREE, g1=[Enable : nr116d]-inst->[Wikipedia]-part->[Edit_Wiki] -
  -obj->[Edit_Page],
  -agt->[Actor];, g2=[Enable : nr116d]-inst->[Wikipedia]-part->
  [Edit_Wiki]-obj->[Edit_Page], g3=[Wikipedia]-part->
  [Edit_Wiki]-obj->[Edit_Page]}
```

The resulting acceptable enabling functionalities for  $fr$  #158 give the specifier two possible functions to implement the requirement: *Edit Page* and *Edit Discussion Page*. It is now up to the specifier to select the appropriate one, based on her experience with the domain. Jane chooses to have the position described on the climate change wiki page itself, not on the discussion page of that page, as the wiki page is what the general public, not interested in

the meta-discussion, will most likely read. She therefore defines the following support definition:

```
[Support : #213]–  
  (Obj) → [Funct_Req : #158]  
  (Inst) → [Wikipedia] → (Part) → [Edit_Wiki] → (Obj) → [Edit_Page]
```

---

## 11.5 Discussion and Conclusion

In this chapter, we examined how to optimize the design of tool systems for collaborative communities using conceptual graphs. Ever more, such communities are supported by rapidly evolving systems of (social) software tools. However, there are many dependencies within and between the tool system and the usage context. Such dependencies can be captured with what we called functionality mappings. Using these mappings, tailored functionality matching processes can be set up that help optimize the design process. Using the power of conceptual graph theory, in particular the properties of graph generalization hierarchies and basic projection operations, many of these matching processes can be semi-automated.

The role and definition of functionality matching processes is not at all trivial. In this chapter, we only gave an illustration of how such processes could be implemented. More sophisticated, complete, and sound approaches are conceivable, but our goal here was modest: introducing a way of conceptual thinking about social software design for collaborative communities, showing the need for functionality matching, and firing an opening shot in the direction how this could be operationalized in practice. Future research should lead to more systematic approaches of conceptualizing, implementing, and applying functionality matching processes in the design of social software systems.

One direction in which this work could be extended is by expanding the conceptualization of the design process which embeds the kind of functionality matching processes introduced here. Work on testbed development for laboratories, such as pragmatic methodologies for knowledge representation tools research and development, could inform such conceptualizations Keeler and Pfeiffer [2006]. Successful collaborative development requires (1) a system architecture and integration to explore ways that people and machines can use component technologies most effectively, (2) a research program to study the conditions required for collaboration, and (3) user-oriented rapid-prototyping testbeds, to understand the impact of the technologies used Science and Board [1993]. Pattern languages can help to capture communication knowledge of large distributed communities Schuler [2002], particularly useful in the design

of collaboratories. In de Moor [2004], we used conceptual graphs to model collaborative improvement as a set of layered testbed processes: information and communication processes enabled by tools, workflows through which the community accomplishes its goals, design processes in which they change their socio-technical system, and improvement processes in which the design processes themselves are made more effective and efficient. We showed how conceptual graphs can be used to optimize these processes. Combining that approach with the detailed functionality matching processes proposed in this chapter, could be one useful way to go about optimizing social software design using conceptual graphs.

---

## **11.6 Student Projects**

### **11.6.1 Project 1**

This chapter introduced a basic concept type hierarchy. Find a number of state-of-the-art research articles on online communities, web services design, etc. Using these articles, refine and expand the hierarchy presented in this chapter. Which newly found concept types are universal, which ones are domain or case-dependent?

### **11.6.2 Project 2**

The functionality mappings presented in this chapter are only very basic. Improve the definitions given. E.g., currently functionality requirements concern individual functions, but probably aggregates of functions could be useful as well. Required implementation-mappings currently concern full tools, but might also be defined at the module or even function-level. What impact do your revised definitions have on the functionality matching processes?

### **11.6.3 Project 3**

Select two cases of (online) collaborative communities, for example e-learning communities, gaming communities, or corporate communities. Using the (revised) concept type hierarchy and functionality mappings introduced in this chapter, try to characterize the socio-technical systems in the respective cases. If necessary, further modify the concept type hierarchy and functionality mappings.

#### 11.6.4 Project 4

The example of a functionality matching process given in this chapter was that of implementing a functionality requirement. However, there are many more applications of such processes. One example not worked out is that of checking software quality aspects, like the effectiveness, efficiency, security, and legitimacy of software (components). Select one such quality aspect and operationalize it in terms of functionality mappings and matching processes. Apply this process to your case description of the previous assignment.

#### 11.6.5 Project 5

Much advanced web services-related research is focusing on standards like the Universal Description, Discovery, and Integration (UDDI) protocol<sup>15</sup>, Business Process Modeling Notation (BPMN)<sup>16</sup> and Business Process Execution Language (BPEL)<sup>17</sup>. Find a project description using these standards and model selected representations and processes using the notions introduced in this paper as a starting point. How could our functionality matching approach help address problems identified in the selected project description?

#### 11.6.6 Project 6

We have used the projection operation for the matching process. What roles could other core CG operations (e.g. `maxJoin`) play in developing new ways of functionality matching and design? How could you use these operations to make the example process presented in this chapter more efficient?

#### 11.6.7 Project 7

Amine is a powerful open source platform for knowledge system implementation<sup>18</sup>, with a strong grounding in conceptual graphs. In particular its Prolog+CG module is very useful for CG representation and reasoning, as it combines a range of CG representations with the reasoning power of Prolog. Make an Amine implementation of the functionality matching process presented in this chapter.

<sup>15</sup><http://www.uddi.org/>

<sup>16</sup><http://www.bpmn.org/>

<sup>17</sup><http://en.wikipedia.org/wiki/BPEL>

<sup>18</sup><http://amine-platform.sourceforge.net/> See <http://www.huminf.aau.dk/cg/> for an online course.