C-Merge: A Tool for Policy-Based Merging of Resource Classifications

Florian Matthes, Claudia Niederée, and Ulrike Steffens

Software Systems Institute,
Technical University Hamburg-Harburg, Hamburg, Germany
{f.matthes,c.niederee,ul.steffens}@tu-harburg.de
www.sts.tu-harburg.de

Abstract. In this paper we present an interactive tool for policy-based merging of resource-classifying networks (RCNs). We motivate our approach by identifying several merge scenarios within organizations and discuss their individual requirements on RCN merge support. The quality-controlled merging of RCNs integrates the contributions from different authors, fostering synergies and the achievement of common goals.
The *C-Merge* tool design is based on a generalized view of the merge process and a simple but flexible model of RCNs. The tool is policy-driven and supports a variable degree of automation. Powerful options for user interaction and expressive change visualization enable substantial user support as well as effective quality control for the merge process.

Keywords: Categorization, Taxonomy, Merging, CSCW, Knowledge Management, Knowledge Visualization, Quality Control

1 Introduction

Effective cooperative construction, structuring, and handling of digital content is a crucial factor in the information society. Beyond the use of digital documents cooperative work with content also involves information resources like personal interests, special expertise etc. Effective discovery and use of, as well as communication about all these resources can be improved by imposing a common classification scheme.

The materialization and adequate visualization of classification hierarchies leads to *resource-classifying networks* (RCN) that use enriched classification hierarchies as an access structure improving information discovery, navigation and exploration. Innovative graphical user interfaces further enhance the usability of such networks.

The construction of RCNs is often a cooperative, long-term effort which includes extension, correction, refocusing as well as restructuring and reacts to new developments and insights assuring a high quality of the classified collection. The construction process is a mix of autonomous efforts, close and loose cooperation as well as online and offline work. This leads to separate, partly competing, partly complementing artifacts that have to be reintegrated to gain a common overall result. RCN merging, thus, is an integral part of the construction process and is required

– to achieve consensus between cooperation partners,
– to benefit from co-workers contributions, and
– to exploit independent, but semantically related evolution.

The potential size and complexity of RCNs makes manual merging a tedious task. Hence, semi-automatic merge support is crucial. This paper presents *C-Merge*, a prototype tool that implements a proposal-oriented approach for merging RCNs. It combines powerful merge support with a user-defined degree of automation and effective options for user intervention.

Flexible RCN merge support is motivated in the next section by considering various cooperative merge scenarios. Section 3 discusses the RCN merge process requirements together with the solutions employed in our approach. The *C-Merge* service architecture and functionality is summarized in section 4. The paper concludes with a discussion of related work and future research directions.

2 Motivation: Merging RCNs

Resource classifying networks play an important role in the cooperative work with information. This section starts with a description of our model of RCNs and identifies several merge scenarios which require flexible RCN merge support.

2.1 A Model for Resource-Classifying Networks

The classification of information objects according to a predefined hierarchy of concepts is an important contribution to their description and discovery. Resources that are structured this way include documents in traditional and digital libraries, co-workers expertise as found in knowledge management, physical facilities like rooms, and events like conferences. If classification hierarchies are materialized they can be used for the structuring and navigation of an information space and contribute to communication and a common domain understanding inside a community [10].

For the merge process we consider a simple but flexible model of such materialized classification structures termed *resource-classifying networks* (RCNs) in what follows. They consist of three integral parts:

– *Classifier nodes* represent classification categories. They comprise the name of the category, an optional ID, a description of the category and further properties.
– *Content nodes* represent the classified resources, which may be local or remote. It is assumed here that resources are identified by a URL. The information resources themselves are not considered part of the RCN.
– Three types of links can exist between the nodes. *Parent links* between classification nodes build up the classification hierarchy, multiple inheritance inclusive. *Classification links* connect content nodes with classification nodes and *jump links* connect arbitrary nodes to represent general associations.

In contrast to formal approaches like ontologies known from AI [5] we consider a restricted set of link types and do not assume a formal description of semantics. Hence, our approach is applicable to more simple, ad-hoc hierarchies as they emerge in many domains and organizations. RCN example applications are traditional classification hierarchies, Web catalogs [7], and knowledge portals [9].

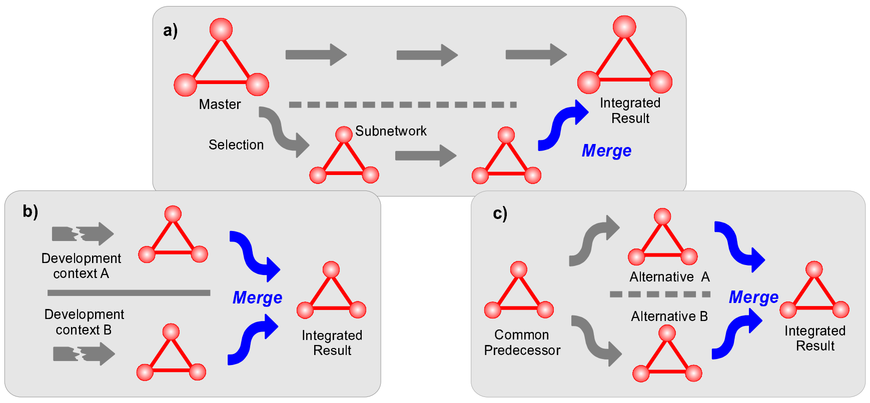


Fig. 1. a) Re-integration of subproject results; b) Merging independent developments; c) Integrating parallel versions

2.2 Merge Scenarios

When developing RCNs the need for merging arises in several cooperative situations:

– A work group is asked to autonomously revise and extend a part of an organization's RCN. The use of autonomous subnetworks resolves the conflict between organizational and work group view [3]: The organization's view is focussed on long-term usability and consensus with the organization's objectives whereas work groups creatively change the RCN according to their task. The revised subnetwork is later re-integrated into the organization's network (see Fig. 1 a)).
– An author discovers an RCN of another domain expert in the Web, which has a focus similar to his own network (see Fig. 1 b)). A controlled integration of the overlapping areas allows the author to benefit from this expert's contributions.
– Two independent project teams work on alternative proposals for the restructuring of an organization's intranet developing two RCNs. The intranet's acceptance can be improved by carefully merging both proposals (see Fig. 1 c)).

These situations differ in their requirements. Adequate merge support depends on factors like the relationship between authors or the importance of the respective RCNs, which influence the degree of automation and the quality control requirements. Confidence in an author's expertise, for example, enables a higher degree of automation whereas integrating developments from an unknown author opts for more quality control. In each case system support has to enable the user to understand the consequences of integrating contributions. A user-friendly visualization, as implemented in the *C-Merge* tool (see Fig. 2), is a major step in this direction.

3 System Analysis and Design

The characteristics of RCNs as well as their crucial role for information structuring and discovery impose special requirements on a process for merging this kind of information objects. This section starts with a general overview of the process of merging which in the second part is adapted to the requirements of our RCN model.

3.1 Overview of the Merge Process

Considered on a general level four phases can be identified for a merge process:

1. *Merge configuration:* The definition of a merge configuration fixes the number and the roles of the information objects involved in the merge process.
2. *Matching:* The computation of a matching identifies the corresponding components of the information objects to be merged.
3. *Change detection:* The differences between the considered information objects are determined by a change detection algorithm.
4. *Change integration:* The contributions are combined into one result object.

Manual merging provides the user with full process control but may be tedious, especially for larger information structures like RCNs. Fully automated merging, on the other hand, is the most comfortable solution, but quality control and conflict resolution completely steered by the system are not always acceptable. Semi-automatic solutions provide a compromise between user-driven control and user comfort.

Merge Configuration This phase assigns roles to the involved information objects, the *merge candidates*: Candidates which contain the contributions to be merged appear in the role of *change sources*. Furthermore, a *change target* is chosen, i.e. a candidate into which the contributions are to be integrated. *Change reference* is the third possible role. A merge candidate in this role is compared with the change source(s) to compute relevant differences. The roles change reference and change target often coincide, but there are other conceivable options, too.

Matching A prerequisite for change detection is the computation of a *matching* identifying corresponding parts in the considered merge candidates. For the matching the merge candidates have to be divided into *components* where a component can e.g. be a sentence in a text file, an element of a collection, or a leaf element in an XML document. The similarity of components depends on different factors:

Component content: Equality or similarity of content is the most obvious evidence for the similarity of two components.

Component context: The similarity of two components may also depend on their environment or context. In a tree, for example, the similarity of two nodes is influenced by the similarity of their children and/or parents.

Component properties: Meta information like size, author or creation date may also be taken into account when computing the similarity of two components.

Component type: In many cases components that are not of the same type are not compared at all. In contrast, some component types may be considerably similar for the respective components to be compared.

Comparing each component of the change source with each component of the change reference results in a large number of comparisons and makes matching inefficient. In many cases it is possible to either exclude components of the change reference from the set of potentially matching candidates or to identify the most promising candidates with little effort. For the computation of the actual matching one or more of the above mentioned similarity factors can be chosen and combined.

Change Detection The purpose of this phase is to determine the change source's contributions to be integrated into the change target. They are represented by a so called *Δ-collection*, a partially ordered collection of change operations from an *operation repertoire*, which, when applied to the change target, adopt the change source's contributions. The Δ-collection is either computed by comparing change source and reference or extracted from a change history. The change detection process is characterized by the chosen operation repertoire and the granularity of considered changes.

Change Integration This phase integrates the identified changes into the change target. An uncontrolled application of the operations from the Δ-collection may reduce the result quality. Therefore, the operations are filtered before they are applied to the change target. The filter process may depend upon user interaction, a merge strategy, consistency rules, or the state of the change target.

Depending on the considered merge scenario and the type of information objects different degrees of automation are adequate for change integration. Semi-automatic merging may support automatic detection and visualization of changes which are manually accepted or rejected by the user (e.g. [14]). An adequate change visualization enables comprehension and reliable integration decisions. A flexible degree of automation is achieved by policy-based merging as discussed in [11] where a merge policy determines what kind of changes are automatically integrated or rejected and what kind of changes require interactive approval.

3.2 Design of the RCN Merge Process

The RCN merge process is influenced by two competing requirements: The complexity of the networks calls for a high degree of process automation to make it feasible. Yet, RCNs are often high investment structures intended for the long-term use making strict quality control crucial. In our approach matching and change detection are automated but also augmented with options for user intervention. This is combined with a flexible degree of automation in the change integration phase, which is of special importance for quality control.

Another issue for semi-automatic RCN merging is the sequence of components to be merged. During the RCN merge process the user must be guided through the network in a way that preserves his orientation to support meaningful integration decisions. Proceeding along the classification hierarchy seems intuitive here.

Our work focusses on the merging of the network structure. For the node content we rely on existing approaches for document merging (e.g. [8]).

Merge Configuration Although merging of more than two networks might be desirable in some organizational contexts (see Sect. 2) and is also technically possible, keeping track of the changes in all the networks would probably overstrain the user. We therefore restricted ourselves to two merge candidates at a time, where one appears as change source and the other as both, change target and reference. This corresponds to the scenario in which an author integrates contributions from another RCN (change source) into his own one (change target and reference).

The coverage of the two networks to be merged may differ substantially. One network may for example structure the entire area of digital library research whereas the second network may be restricted to IR issues. This situation is handled by providing support for the merging of user-defined subnetworks.

RCN Matching The nodes within RCNs carry an elaborate and often also stable part of the network's semantics. Using RCN nodes as matching components is therefore a straightforward approach.

Content and classifier nodes represent different types of RCN components. As they perform different tasks within the RCN, instances of different types are disregarded for matching. Considering node content, the matching of classifier nodes mainly relies on the name of the corresponding concept and can also involve the ID or description if present. Content nodes are matched by comparing the referenced resources. The matching result is further refined by taking into account the nodes' context, mainly focussing on the comparison of parent and classification links.

To achieve a tolerable efficiency for the RCN matching process, it is subdivided into two phases. Initial matching candidates are computed making use of indexes over the node names and the resource's URLs, respectively. The intermediate result set is then further narrowed by comparing the nodes' context.

A comparison of the computed similarities with two customizable threshold values subdivides the matching pairs into proposed and confirmed pairs. Pairs with a similarity below the smaller threshold are not considered as a matching at all.

Change Detection in RCNs Taking a node-centered approach change detection and change integration are realized as alternating phases within the merge process.

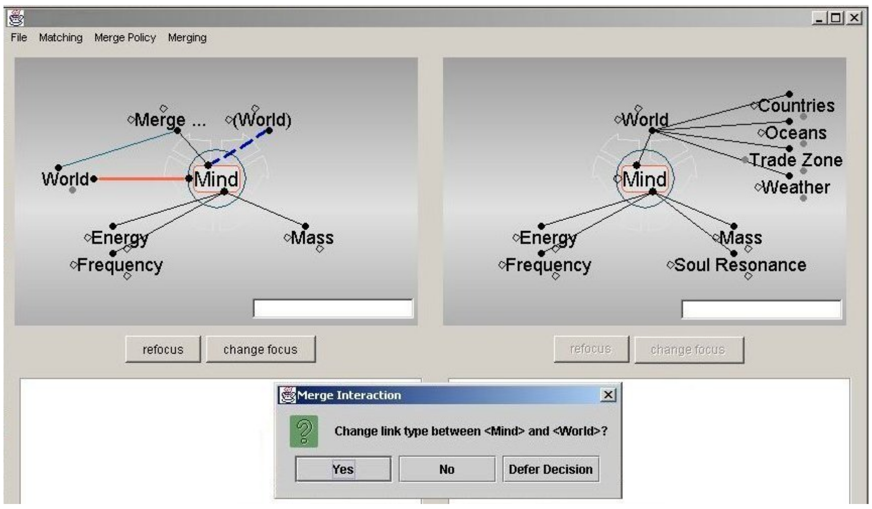
The operation repertoire for networks consists of operations on nodes and links. Our approach supports a restricted repertoire that is manageable for the user. It includes operations for node insertion, update, and deletion as well as for link creation and removal. Additionally, we consider two operations for link update: *re-parent*, redirecting a parent link to a different parent node, which corresponds to moving a subtree, and *changeType*, turning a parent link into a jump link or vice versa.

A precomputation of the complete Δ-collection in a separate pass is inadequate for RCNs. It does not take dynamic changes during merging into account (see Sect. 4.2). Instead our approach locally determines the differences node by node, just in time for change integration. Change detection and integration rely on the notion of a *focus node* which is the node currently processed. In succession, each node becomes a focus node in a variant of a breadth-first processing order along the hierarchy.

For change detection the focus node environment is compared with the environment of its matching partner. The environment of a node includes all direct neighbors of a node and the connecting links. Links that differ within the two environments are matched with a set of possible difference situations in order to identify the operations for the Δ-collection. Single nodes, i.e. nodes that have no matching partner, get special treatment.

Change Integration in RCNs The automation of the change integration phase is a challenging task. For RCNs this is further tightened by the complexity of the information structure to be merged. To achieve a variable degree of interactivity we decided to employ merge policies as proposed in [11].

A merge policy customizes the merge process by defining rules for handling occurring changes. Since we have only one change source, the policies can be implemented by simplified, single-column merge matrices that provide an entry for each possible change operation. This entry specifies if the operation is automatically applied, ignored or if the user is asked for a decision. The matrices are used to look up the further proceeding for the operations found in the Δ-collection. Flexibility is increased by enabling users to define their own merge policies.



Change link type between <Mind> and <World>?

Fig. 2. Proposal of Changing a Link Type

Most policies chosen for RCN merging will not be fully automatic: A change will be proposed to the user and he decides about its integration. The visualization must enable the user to understand the proposed change as well as the consequences of its acceptance or rejection. For this purpose the change is presented as part of the focus node context: The change under consideration is entered into the focus node environment and highlighted according to the type of change. As an example figure 2 shows the proposal of changing a link type. Via a dialog box the user can accept or reject the change. In addition we enable intermediate browsing through the RCN during change integration so that the user can gain more context information.

4 The *C-Merge* Prototype

This section presents *C-Merge*, a flexible Java prototype tool for the merging of RCNs implemented at our department. The implementation relies on existing components and libraries and uses a commercially available RCN format. The prototype is characterized by a proposal-based change integration and offers several options for user interaction and intervention enabling a flexible process control.

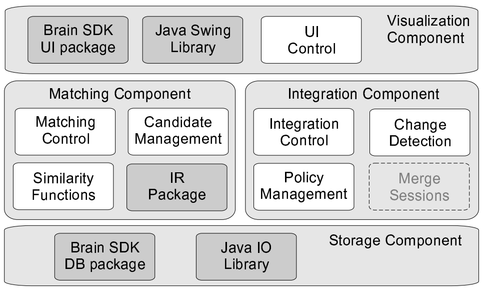


Fig. 3. The *C-Merge* Architecture

4.1 System Overview

***C-Merge* Component Architecture** The tool consists of four main components (see Fig. 3) implementing the approaches presented in the previous section:

The **integration component** implements the presented approach for change detection, provides support for the definition, management and application of merge policies and controls the change integration process according to the chosen policy. Advanced merge session support is planned for a future version.

The **matching component** computes the RCN matching. It includes a component for the management of matching candidates, matching pairs and single lists as well as a component for the control of the matching process. An existing Java IR package is employed to compute the similarity of text properties.

The **visualization component** relies on the Java Swing library and the UI package of the *BrainSDK*, a Java library that comes with the RCN format employed in the prototype. This imported functionality is integrated and controlled by a set of application specific user interface classes.

The **storage component** uses the *BrainSDK* DB package functionality for the persistent storage of the employed RCN format. Further merge-related information like merge policies are stored using the classes of the Java IO library.

Processing Sequence The UML activity diagram in figure 4 shows a pass through a typical merge session with the *C-Merge* prototype. During the process intermediate merge results can be stored.

The *Brain* RCN format The prototype uses a commercially available RCN format, namely the *Brain* format from *Natrificial* (www.thebrain.com). The metaphor behind this format is a brain consisting of a set of associated thoughts. The *thoughts*, which are the nodes of the *Brain* RCNs, are connected by two types of links: *parent links* create a hierarchy and *jump links* express general references between thoughts.

The *BrainSDK*, a development kit provided by Natrificial, includes Java class libraries for the visualization, manipulation, and storage of the Brain RCNs and provided a good starting point for the implementation of our tool.

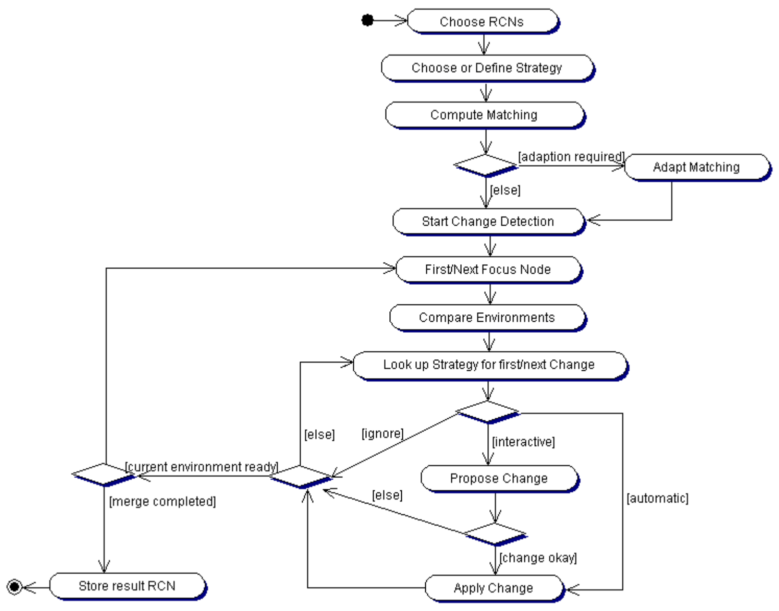


Fig. 4. A typical *C-Merge* session

4.2 User Interaction and User Intervention

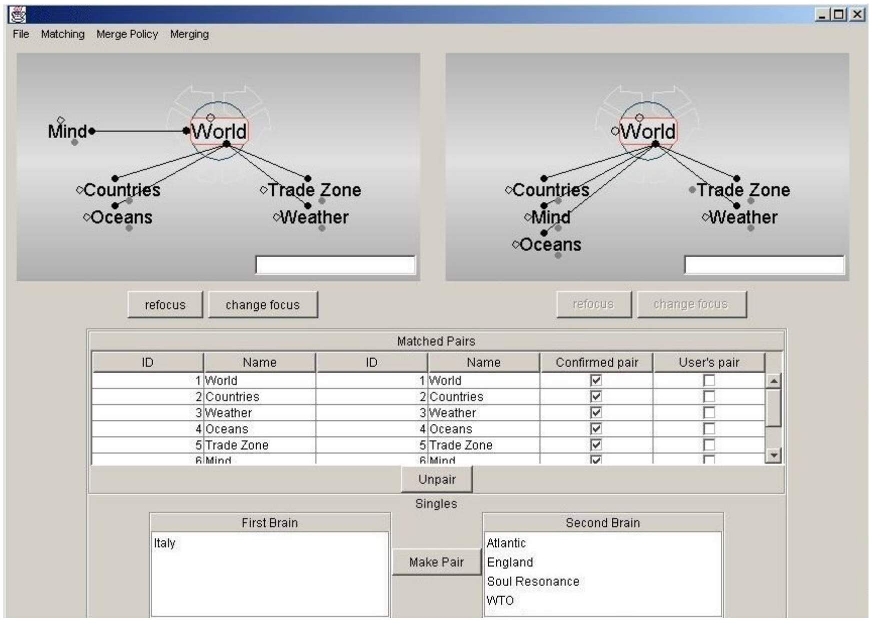
A challenging issue in tool design is the coordination of user interaction with the automatic steps of the merge process. In addition to *user input* at well-defined points like merge policy definition and integration decisions as required by the policy there is another kind of user interaction: Task-driven *user intervention* spontaneously manipulating and redirecting the process flow. Possible user interventions are:

– Update of the matching
– Change of the focus node
– Update of the RCN

The increased complexity of the control flow requires additional book-keeping. Node processing states are used for this purpose. The states *unprocessed*, *in work*, and *ready* are distinguished. An additional state, *modified*, marks nodes that have already been processed but need reconsideration because of some user intervention.

Change of Focus Node The predefined processing order for change detection and integration may be changed by manually choosing a new focus node. The user may decide interactively if he wants only this node processed or the entire subtree rooted at the new focus node. For the subtree option the processing of the actual focus node f_{act} is considered completed. The node processing state changes from *in work* to *ready*. The chosen node f_{new} becomes the new focus node and is processed next. Subsequently, the subtree of which f_{new} is the root is processed. Finally, processing returns to the normal order. A nested change of focus node is possible.

During the merge process the user can browse both RCNs to get more context information. Pressing a button he can return to the current focus node.



Matched Pairs

ID	Name	ID	Name	Confirmed pair	User's pair
1	World	1	World	☑	☐
2	Countries	2	Countries	☑	☐
3	Weather	3	Weather	☑	☐
4	Oceans	4	Oceans	☑	☐
5	Trade Zone	5	Trade Zone	☑	☐
6	Mind	6	Mind	☑	☐

Singles

First Brain	Second Brain
Italy	Atlantic
	England
	Soul Resonance
	WTO

Fig. 5. *C-Merge* User Interface

Update of Matching The computed matching results in a list of matched nodes presented side by side, where some pairs are marked as confirmed matchings. In addition there is a list of single nodes for each involved RCN (see Fig. 5). The matching can be modified manually by the user which can have implications for the rest of the matching as well as for the change detection process.

The effects on change detection are handled through the node states. If a matching is changed all involved nodes in state *ready* are changed to the state *modified* and change detection re-starts at the top. Only nodes that are in the state *in work*, *unprocessed* or *modified* are considered in this pass through the RCN.

No automatic re-matching is performed as an effect of a manual change, unless explicitly triggered by the user, because, although useful in some settings, it may effect large parts of the network and complicate the running merge process.

Update of the RCN Merging RCNs implicitly involves rethinking of the networks' semantics. To directly integrate new ideas the user is allowed to change the RCN acting as change target in the course of the merge process.

RCN modifications influence the merge process, especially change detection. The consequences of such changes depend on the operation type and the states of the involved nodes. For an *unprocessed* node a modification of it or its environment will be handled as part of the normal processing. For a node that is in state *ready* a modification is considered as a post-merging operation that requires no further processing. For a node that is currently *in work* change detection is reconsidered taking into account the integration decisions already made by the user.

5 Related Work

Like RCNs, topic maps [2] are materialized classification structures, but exhibit a richer meta level with typed nodes and links as well as scopes for topics. Ontologies [5] used for the conceptualization and representation of knowledge are based on a more formal approach. RCNs can be considered as weakly structured ontologies [6] that exhibit a restricted set of link types and do not formalize the semantics.

The need for merging comes up in cooperation situations where stronger forms of synchronization are not possible. Examples are offline work, parallel work due to time constraints and loose cooperation with high autonomy for the cooperation partners. These forms of so-called *autonomous collaboration* are motivated and discussed in more detail in [4]. Further mechanisms relevant in such cooperation contexts are change notification (e.g. [12]) and advanced link management [13].

Merging support mainly exists in multi-version working contexts like cooperative software development and information artifact authoring. Hence, typical merge candidate formats are pure text files [8], other document formats like Microsoft Word, and source code files [15]. The merge candidates considered in our tool are often high-investment structures whose correctness and adequateness plays an important role for an organization. This imposes additional requirements on merge support.

All existing merge tools are semi-automatic where a frequent solution is automatic change detection and visualization combined with interactive change integration (e.g. [8]). Our tool is based on a more flexible approach proposed in [11], where the degree of automation can be gradually adapted via merge policies. A somewhat different approach to merging is taken in the GINA framework [1], which is based on protocolled changes managed in operation histories. Special redo operations enable operation re-application in a modified object state during merging.

6 Conclusions and Future Work

In this report we presented *C-Merge*, a flexible tool for the semi-automatic merging of resource-classifying networks. First experiments with the tool at our department showed that it enables comfortable and quality-preserving merging of RCNs. Especially the enhanced change proposal visualization and the customizable merge policies contributed to user satisfaction.

Merging larger RCNs can be a time-consuming task even with semi-automatic merge support as provided by our tool. Thus, it is desirable to have persistent merge sessions that can be interrupted and resumed later without losing the effort already invested. In addition to the current state of the change target RCN further process information has to be made persistent for this purpose. We plan to integrate a merge session management into the prototype.

In the current prototype we restricted ourselves to a simple operation repertoire avoiding information overload in the visualization. In a future version we will experiment with the detection and visualization of important more complex change operations, which are typical for RCN restructuring, like the splitting of a node or the merging of two nodes. We expect that a carefully tuned operation repertoire provides the user with valuable additional information for his integration decisions.

Some merge scenarios, as e.g. the integration of parallel versions of a common predecessor, are more adequately mapped by a three-way merge. For this reason we intend to examine options for three network merge configurations although the danger of information overload is rather high. A careful user interface design and conflict management are crucial in this context.

Acknowledgments: The described research was partly supported by the HSPIII Project WEL and the DFG Project KOLIBRI (DFG Schm450/7-1, MA2005/1-2).

We would like to thank Siripong Treetasanatavorn and Hendry Chandra for their help in implementing the prototype.

References

1. Thomas Berlage and Andreas Genau. A Framework for Shared Applications with a Replicated Architecture. In *Proceedings of the ACM Symposium on User Interface Software and Technology, Atlanta, GA*, pages 249–257, November 1993.
2. Michel Biezunski, Martin Bryan, and Steve Newcomb. ISO/IEC FCD 13250:1999 - Topic Maps, April 1999. http://www.ornl.gov/sgml/sc34/document/0058.htm.
3. Giorgio De Michelis, Eric Dubois, Matthias Jarke, Florian Matthes, John Mylopoulos, Joachim W. Schmidt, Carson Woo, and Eric Yu. A Three-Faceted View of Information Systems. *Communications of the ACM*, 41(12):64–70, December 1998.
4. W. Keith Edwards and Elizabeth D. Mynatt. Timewarp: Techniques for Autonomous Collaboration. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI'97), Atlanta, GA*, pages 218–225, March 1997.
5. T. R. Gruber. A translation approach to portable ontology specifications. Technical Report KSL 92-71, Computer Science Department, Stanford University, CA, 1993.
6. Michiaki Iwazume, Kengo Shirakami, Kazuaki Hatadani, Hideaki Takeda, and Toyoaki Nishida. IICA: An Ontology-based Internet Navigation System. In *AAAI Workshop Internet-Based Information Systems, Portland, OR*, pages 65 – 78, August 1996.
7. Yannis Labrou and Tim Finin. Yahoo! As an Ontology: Using Yahoo! Categories to Describe Documents. In *Proceedings of the 8th International Conference on Information Knowledgement (CIKM-99)*, pages 180–187, N.Y., November 2000.
8. David MacKenzie, Paul Eggert, and Richard Stallman. Comparing and Merging Files. http://www.gnu.org/manual/diffutils-2.7/html_mono/diff.html, September 1993.
9. F. Matthes and U. Steffens. Establishing a Cooperative Digital Library for Teaching Materials - A Case Study. Technical report, Software Systems Group, Hamburg University of Technology, Germany, August 2000.
10. Rainer Müller, Claudia Niederée, and Joachim W. Schmidt. Design Principles for Internet Community Information Gateways: MARINFO - A Case Study for a Maritime Information Infrastructure. In *Proceedings of the 1st International Conference on Computer Applications and Information Technology in the Maritime Industries (COMPIT 2000), Potsdam/Berlin, Germany*, pages 302–322, April 2000.
11. J. Munson and P. Dewan. A Flexible Object Merging Framework. In *Proceedings of the ACM CSCW'94 Conference on Computer Supported Cooperative Work, Chapel Hill, NC*, pages 231 – 242, October 1994.
12. NetMind. Mind-It Notification Service. http://www.netmind.com/.
13. Claudia Niederée, Ulrike Steffens, Joachim W. Schmidt, and Florian Matthes. Aging Links. In *Research and Advanced Technology for Digital Libraries, Proceedings of the 3rd Europ. Conf., ECDL2000, Lisbon, Portugal*, pages 269 – 279, September 2000.
14. Presto Soft. Exam Diff Pro. http://www.nisnevich.com/examdiff/examdiffpro.htm.
15. Bernhard Westfechtel. Structure-Oriented Merging of Revisions of Software Documents. In *Proceedings of the 3rd International Workshop on Software Configuration Management*, pages 68 – 80, 1991.