Semi-automatic Merging of Content Networks: Policy-based Customization

Master's Degree Project Work

submitted to
Dipl.-Inform. Claudia Niederée and Prof. Dr. Florian Matthes

composed by
Siripong Treetasanatavorn
Master Program in Information and Communication Systems
Technical University Hamburg–Harburg

25th June 2001

Contents

1 Introduction **3**
- 1.1 Project Goals . . . 4
- 1.2 Report Structure . . . 4

2 Merge Process Introduction **5**
- 2.1 Cooperative Merge Scenarios . . . 5
  - 2.1.1 Merging Parallel Development . . . 5
  - 2.1.2 Merging Independent Development . . . 5
  - 2.1.3 Reintegration of Subproject Results . . . 6
- 2.2 Merge Process Overview . . . 6
  - 2.2.1 Merge Configuration . . . 6
  - 2.2.2 Matching . . . 7
  - 2.2.3 Change Detection . . . 7
  - 2.2.4 Change Integration . . . 7
- 2.3 Merge Policies . . . 7
  - 2.3.1 Overview . . . 8
  - 2.3.2 Consolidation Merge Policy . . . 8
  - 2.3.3 Reconciliation Merge Policy . . . 8
  - 2.3.4 User-defined Merge Policy . . . 8
- 2.4 Related Work . . . 9
  - 2.4.1 UNIX diff3 . . . 9
  - 2.4.2 GINA: A Replicated Architecture Framework . . . 9
  - 2.4.3 A Flexible Object Merging Framework . . . 9
- 2.5 Resource-Classifying Networks (RCN) . . . 9

3 System Design Process **11**
- 3.1 Central Design Issues . . . 11
  - 3.1.1 Merging Resource-Classifying Networks . . . 11
  - 3.1.2 Level of User Interaction . . . 12
- 3.2 Merge Algorithm Design . . . 12
  - 3.2.1 Merge Process Cycle . . . 12
  - 3.2.2 Merge Sequence Generation: Breadth-First Algorithm . . . 13
  - 3.2.3 Merging Process States . . . 15
- 3.3 Change Detection and Change Integration . . . 16
  - 3.3.1 Change Detection Algorithms . . . 17
  - 3.3.2 Change Integration and Visualization . . . 22

3.3.3 Merging States 23
3.3.4 Merge Session 24

4 Employed Technologies **25**
4.1 Java 25
4.2 TheBrain 25
4.2.1 RCN Format Requirements 25
4.2.2 The Brain RCN format 26
4.2.3 The BrainSDK 27

5 Prototype Functions and Graphical User Interfaces **29**
5.1 Prototype Functions 29
5.2 Prototype Architecture 29
5.3 User Interface and User Interaction 30
5.3.1 User Interface Overview 30
5.3.2 User Interaction and User Intervention 31
5.3.3 Visualization of Proposed Change 32
5.3.4 Merge Policy Characteristics 34

6 Conclusion **35**

List of Figures

2.1 Merging Parallel Development Scenario 6
2.2 Merging Independent Development Scenario 7
2.3 Reintegration of Subproject Scenario 8

3.1 Activity Diagram of Main Merge Process 13
3.2 State Diagram of Merging Process States 16
3.3 Change Detection Process 17
3.4 Change Integration with Consolidation Merge Policy 20
3.5 Change Integration with Reconciliation Merge Policy 21
3.6 State Diagram of Merging States 23

4.1 An Example of Resource-Classifying Network: TheBrain 26

5.1 Prototype Architecture 30
5.2 Prototype User Interface Overview 31
5.3 Visualization of Proposed Change 33

Abstract

Effective cooperative construction, structuring, and handling of digital content is a crucial factor in the information society. The organization of content into classification hierarchies improves accessibility, navigation and iterative exploration of relevant topics. Innovative graphical user interfaces further enhance the usability of such content networks. The construction of such content networks is not a single-author, short-term activity but a cooperative, long-term effort including extension, correction and refocusing as well as regular restructuring. The construction effort is a mix of close and loose cooperation, cooperative and autonomous work 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. The merging of information artifacts, thus, is an integral part of the construction process for content networks. Since manual merging is a tedious task, semi-automatic system support is crucial. The goal of this student project is a merge tool prototype that supports a customizable semi-automatic merge process for resource-classifying network (RCN). The focus is on the integration phase of the merge process. Merge policies are used to achieve a variable degree of automation for the change integration phase. Starting from existing research and tools on merging for different data formats like text files and on merge policies, the applicability and adaptation of these approaches to RCNs are examined. The practical part of the student project consists of the cooperative construction of a merge tool for an existing RCN format, integrating the findings on merge policies.

Chapter 1

Introduction

Over the past years large amounts of heterogeneous digital information are proliferating in the information society. The effective constructing, organizing, storing, retrieving, manipulating, and evolving of digital information are therefore crucial tasks, so as to appropriately deal with the information requirement of changing processes and environments, as well as to effectively create new types of information contents which suit the changing conditions. The construction of information content is nowadays less likely to be a single-author and short-term activity, but a cooperative and long-term effort among groups of contributors becomes more commonplace. In addition, each contributor usually holds different, and possibly changing, roles in such an activity. The effective discovery and the use of complex information resources can be improved by imposing a common classification scheme. The adequate and effective visualization of classification hierarchies leads to the concept of *resource-classifying networks* (RCN), which improve information discovery, navigation, and iterative exploration of relevant topics.

As a result, the special concerns to collect new relevant contributions, while maintaining a consistency and an overall direction of the final revision, are of high interest. Additionally, the content construction is typically a mix of close and loose collaborations, and it thus leads to the need for adjustible tools to handle with different situations effectively. Merging tools, which can be applied to integrate the different versions to achieve a common overall result, are particularly useful and attractive.

Tools which facilitate the information construction and integration become inevitably crucial. Merging tools play an important role in a cooperative environment because they fulfill the following significant purposes.

- to achieve consensus between the cooperation partners
- to benefit from cooperation partners' contributions
- to enable and exploit independent and parallel developments

Therefore, it is the goal of this project to develop a software prototype which can provide a customizable degree of automation and interactivity in the merge of the RCNs. The goals of the project are discussed in more detail in the following subsection.

1.1 Project Goals

The main goal of the project is to develop a prototype software application for merging RCNs. In order to support different cooperative merge scenarios, this software prototype will support a customizable semi-automatic merge feature via merge policy customization. It is aimed to be embedded into a larger framework providing a user-friendly environment for the cooperative web-based construction and navigation of know-how portals. For the user-friendly visualization a commercial RCN construction kit is used in the construction of the prototype. In summary, the software prototype is characterized by the following points.

- proposal-oriented approach for change integration
- user-defined degree of automation
- effective options for user intervention
- flexible control of the matching and merge configuration

1.2 Report Structure

The report is structured as follows. Chapter 2 presents an overview of the merge process: cooperative merge scenarios, an abstract merge process overview, the concept of merge policies, related work on merging, and the concept of RCNs. Chapter 3 describes the prototype software development process covering all relevant issues, for instance, the main prototype activity diagram, a discussion of the merge sequence, and the applied algorithms. Before discussing the prototype functions and implementation issues, chapter 4 captures the main employed technologies, Java, the well-known object-oriented programming language, and the development tool for constructing and navigating the RCN, called *BrainSDK*. Thereafter, chapter 5 mainly discusses the prototype functions and the graphical user interfaces. It also includes the issue of the merge matrix implementing merge policies, which are used to customize the degree of the automation and interactivity. Finally, the report concludes with Chapter 6 summarizing the results and discussing some future work.

Chapter 2

Merge Process Introduction

In this chapter, the author describes typical cooperative scenarios that profit from merge support. Thereafter an overview of the merge process is given on a conceptual level. This merge process view is used as a basis for the description of existing merge approaches in this chapter and for the design of the merge prototype in chapter 3. This chapter concludes with an introduction of the idea of the resource-classifying networks, which are to be merged by the prototype.

2.1 Cooperative Merge Scenarios

The construction of content networks is a long-term effort and a mixture of cooperative and autonomous development, the requirements for merging potentially conflicting changes and for collecting relevant contributions from different participating versions become obviously of interests. To give the reader some examples of this description, the author would like to discuss these following scenarios which are considered as main driving forces and are benefited from the merge process.

2.1.1 Merging Parallel Development

The merging *parallel development scenario* (see figure 2.1) assumes that cooperation partners originally share a common predecessor version. According to some necessities, the cooperation partners are forced to develop separate versions of the information object autonomously and in parallel. One example situation is when the project deadline comes close, and it is impossible to finish the project in time in a sequential fashion. Parallel development becomes necessary. However, a single meaningful consolidated version which incorporates contributions from all authors is considered as the main goal.

2.1.2 Merging Independent Development

Another practical scenario is independent developments from different authors (see figure 2.2), sharing a common focus but no common predecessor. It is typical that the merge process in this case becomes much more complex, since numorous conflicts might arise. Nevertheless, the final integrated version should incorporate the contributions which all partners agree upon. This motivates the developments of a flexible and semi-automatic merge tool as proposed in this work.

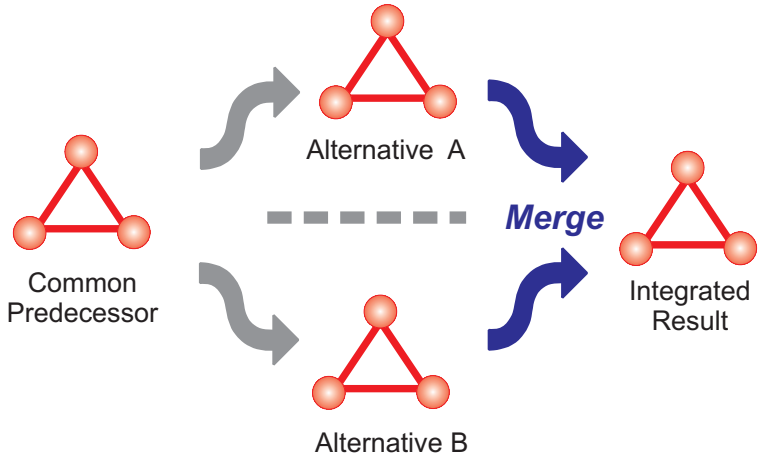

Figure 2.1: Merging Parallel Development Scenario

2.1.3 Reintegration of Subproject Results

The last scenario is settled in the development of a large project, which contains a set of the subprojects. At some point of the development phase, the considered subprojects are split and developed in parallel with the main project. However, at the end of each subproject, the integration of the subproject to the main project is necessary. After merging, the new master project is characterized by the developments of the main project and the specializations found in subprojects. (see figure 2.3)

2.2 Merge Process Overview

The main goal of the merge process is to integrate contributions from different sources or authors to produce a new meaningful merged version. The merge process on objects usually consists of two phases, namely the phase which focuses on determining object equivalence (object matching), and the phase which focuses on merging the matched objects [7]. This approach takes a more fine-grained view. The merge process phases are distinguished as follows.

- Merge configuration identification
- Matching execution
- Change detection
- Change integration

2.2.1 Merge Configuration

At the beginning of the process, the user has to identify the roles of the objects to be merged All of the involved information objects, named merge candidates, should first be assigned roles for the merge process. The object whose contributions are to be integrated takes the role of the *change source*. In order to determine the contributions the change source has to

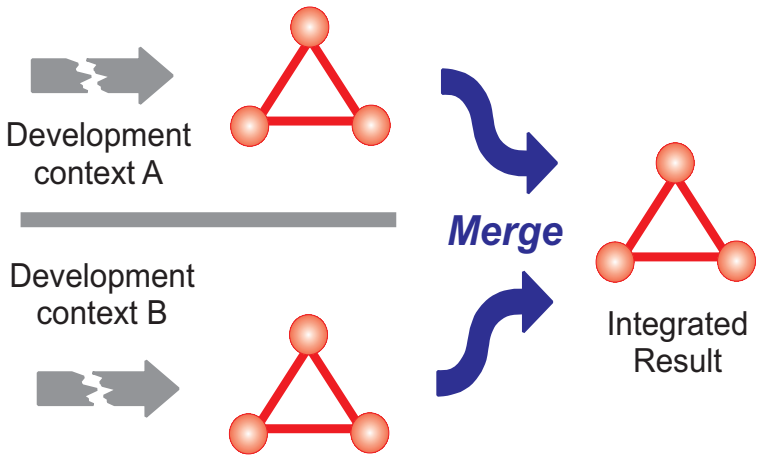

Figure 2.2: Merging Independent Development Scenario

be compared with the object which acts as *change references*. The identified differences are integrated into the object which acts as *change target*. The role of change target and change reference often coincide.

2.2.2 Matching

After defining the merge configuration, a matching pairs of similar component is computed in the second step. The prototype identifies matching pair candidates from the chosen change sources and change references, by applying similarity functions, and it decides about matching based on matching thresholds. The reader can find the detailed description of the matching process in another project, which is conducted by another student project.

2.2.3 Change Detection

After the prototype receives the result of the matching process, it then performs the change detection process. The change detection process consists of a set of algorithms to calculate differences between change reference and change sources. According to the matching result set, the system identifies the corresponding matching pair, and performs the difference calculation to their contents and the environments of the matching pair. Finally, a set of operations is collected by the delta collection process.

2.2.4 Change Integration

From a set of operations received from the previous step, the system is now able to integrate each of those to the change target.

2.3 Merge Policies

From the scenarios described above, it becomes clear that high flexibility is required from a merge support.

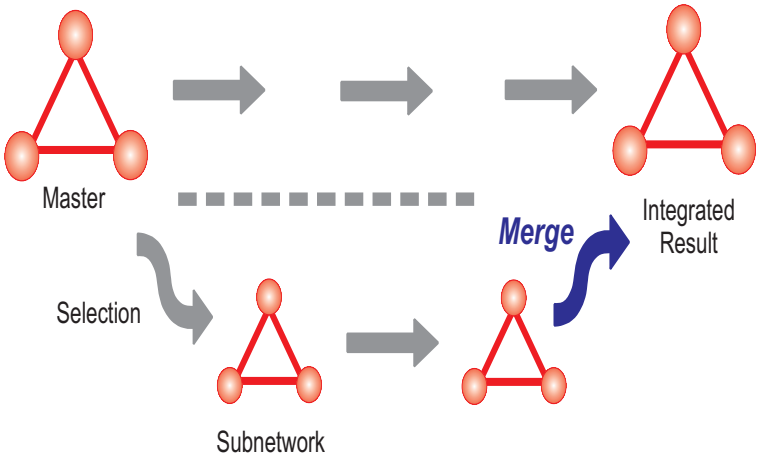

Figure 2.3: Reintegration of Subproject Scenario

2.3.1 Overview

According to the different assumptions on the different scenarios, the *merge policy* [5] is therefore defined to appropriately capture all of the user preferences in order to produce merge results with preferred degree of user interaction. It enables variable degree of automation and interaction in the merge process.

2.3.2 Consolidation Merge Policy

As an example of a merge policy, the *consolidation merge policy* is applied when the merging *parallel development scenario* is considered. Assuming that all of the cooperative partners share a common predecessor version. There are no significant conflicts among the merge candidates, and for most of the parts, the changes are complementary. A consolidation merge automatically integrates all non-overlapping changes.

2.3.3 Reconciliation Merge Policy

Contrary to the *consolidation merge policy*, the *reconciliation merge policy* aims to collect contributions from input versions, which are expected to contain considerable conflicts. This policy is adequate for merging *independent development scenario* with no common predecessor. This merge procedure includes only those parts that all partners agree upon. User interaction is essential in this merge policy.

2.3.4 User-defined Merge Policy

Apart from two mentioned merge policies various other merge policies are discussed in [5]. There are still other situations that require to arbitrarily configure the degree of interactivity. Therefore, the prototype should also offer customizable merge policies.

2.4 Related Work

In this section, some existing merge tools and approaches are described. The issues to be discussed are how other comparable works are created and what the relating advantages, disadvantages, and limitations of those merge tools are.

2.4.1 UNIX diff3

The UNIX diff3 program [4] compares three files, and then produces a script which can generate a merge result file collecting all of the changes of the other two files to the file which that script is applied to. This diff3 is based on the diff program, which can compare and calculate differences between two files. While this program is not limited to the merge which is composed of three merge candidates, the program diff3 is limited the applicability only to merge candidates of type text.

2.4.2 GINA: A Replicated Architecture Framework

GINA [1] is a collaboration tool which has a different approach to the merge process. The GINA framework is based on *command histories* and is a replicated architecture. It maintains the state consistent to other participating applications via *exchanging commands*. Each application maintains a history list of the command objects, which it received from other participating applications. The history lists and an adequate implementation of undo and redo operation are essentially the basis of GINA's unlimited *undo and redo capabilities*. Both features together enable the users to reach every previous state. The history is represented as a tree of command objects. A merge command is performed by sequentializing that two parallel commands by using selective redo, or by taking a series of commands of one branch and then applying to the other end of the other branch. Additionally, the mentioned merge procedure is independent of the commands in the history. In conclusion, the GINA framework offers following important properties, that is, it offers true generality with respect to the object merge functionality, and it also offers automatic differencing and interactive merging.

2.4.3 A Flexible Object Merging Framework

The *flexible object merging framework* [5] holds an important advantage, which is the capability to merge arbitrary object type according to the object model underlying in the paper. It also includes an automatization and a merge policy customization feature. The framework has following properties: automatic differencing, interactive merging, general objects applicability, conflict resolution, as well as customizable and fine-grained configuration of merge policy, which is the key advantage of the approach. The merge matrix is a proposed method to realize configurable merge policies. The customizable *merge matrix* is the central mechanism to solve the problems and conflicts. Each matrix element in the merge matrix has one special action, which indicates how to perform change integration.

2.5 Resource-Classifying Networks (RCN)

In order to improve the information retrieval and accessibility feature, resource-classifying network is an attractive approach. It organizes a collection of content into a classification

hierarchy. With the innovative graphical user interfaces, it also yields the clear structure of the relationships of the given set of information objects, which can also help describe any possible ingrained knowledge as well. For the merge process we consider a simple but flexible model of such materialized classification structures termed *resource-classifying networks* (RCNs). They consist of three parts:

- *Classifier nodes* represent classification categories. They consist of the name of the category and an optional ID, a description for 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 URLs. The information resources are not considered part of the RCN.
- Three types of links can exist between the nodes. *Parent links* between classifier nodes build up the classification hierarchy, *classification links* connect content nodes with classification nodes and *jump links* connect arbitrary nodes to represent general associations.

Chapter 3

System Design Process

Using the ideas from the general merge process covered in the last chapter, a prototype for merging resource-classifying networks (RCNs) is designed. This chapter describes the requirements and design solutions of merging RCNs. Additionally, this chapter provides a discussion of the problems caused by different merge scenarios. The design of the RCN merge tool is the basis for the tool implementation which is described in Chapter 5.

3.1 Central Design Issues

The task to construct a service, which is able to merge a complex class of information objects, namely resource-classifying networks, is accompanied by special requirements for the different steps of the merge process. Furthermore, the prototype has to be applicable in different cooperation situations (as mentioned in Chapter 2) and has to support various degrees of user interaction according to the considered scenario and the user preferences. These requirements improve the applicability of the prototype, but imply higher algorithm complexity in the implementation.

3.1.1 Merging Resource-Classifying Networks

Before identifying design issues for each merge process phase, this section introduces some necessary aspects of the merge process.

Merging Components in Resource-Classifying Networks

According to the specific structure of the resource-classifying networks, the merge task can be further divided into the following subtasks:

- *Content Node Merging*: The merge process should be able to decide whether there is a need to perform a merge of the node content. Generally, the change detection algorithm should be able to detect the differences between information resources.

- *Classifier Node Merging*: This task is similar to the merge task of the content nodes. Typically, the classifier node contains only short text information. Therefore the merge of classifier nodes is considered as a simplification of the content node merge.

- *Classification Hierarchy Merging*: The merge of the classification hierarchy is the integral part merging resource-classifying networks. This task consists of node matching process, environment difference calculation, and change integration.

3.1.2 Level of User Interaction

According to the different cooperation scenarios and the objectives of this prototype, the configurable degree of user interaction is a central design issue. The use of merge policy enables user interaction to all available operations. For each type of change three user interaction degrees are configurable, namely:

- *Automatic*: The considered operation will be integrated automatically without asking for user confirmation.
- *Ignored*: The considered operation will be ignored without asking for user confirmation.
- *Users*: Before the considered operation is performed, the prototype will ask the user for a decision. If he is not ready to make the decision at that moment, the prototype also provides an option to postpone the decision.

3.2 Merge Algorithm Design

In this section, the features of the prototype are described. The first part covers the main cycle of the application, which is explained with the help of an activity diagram. Secondly, an important module is introduced (*merge sequence generation*). It defines the sequence in which the user is guided through the RCN during merging . Thirdly, the *merging process states*will be explained. This part describes the opportunity of the user for the manual control of the module. Subsequently, the combination of change detection and change integration processes are described. The section concludes with the node processing state and the merge session maintenance.

3.2.1 Merge Process Cycle

As shown in the activiy diagram (see figure 3.1), the overall merge process cycle is divided into the following steps.

- *Merge Process Initialization*: This step consists of identifying merge candidates, creating a set of backup copies of merge candidates, and identifying merge policy.
- *Matching*: It computes a set of matching pairs. This part is described in another project [2].
- *Focus Node Selection and Matching Pair Identification*: In the merge process on a set of merge candidates, the prototype divides the entire process into a set of smaller iterations. In each iteration, the prototype first selects a *focus node*. The focus node identifies the current merge focus. Based on this focus node selection, the prototype can identify the corresponding matching partner.

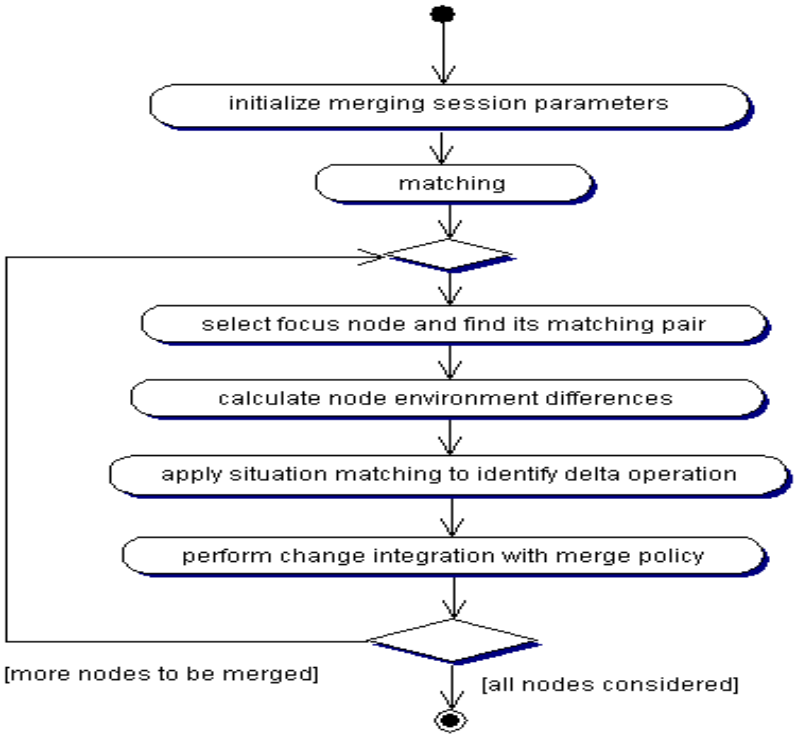

Figure 3.1: Activity Diagram of Main Merge Process

- *Change Detection*: This step performs the change detection algorithms to find the differences between the environments of a given node pair. The differences calculated from this step are either content differences or environment differences. For the latter category, the prototype further identifies necessary changes to the change target. In this step, environment differences are considered as a set of inputs to the algorithm for calculating the operations to be integrated.

- *Change Visualizations and Integration*: This step sequentializes all operations computed by the change detection step, and performs the integration algorithms to the change target according to the merge policy. Depending on the policy an operation is either applied automatically, ignored, or the user is asked for a decision. For user interaction the changes have to be adequately visualized.

- *Termination*: After applying the change integration algorithms to a matching node pair, the prototype determines whether an entire merging process is complete. If not, the iteration will be repeated, which includes selecting the focus node, identifying the matching pair, performing a change detection, and performing a change integration, until the end of entire process is reached.

3.2.2 Merge Sequence Generation: Breadth-First Algorithm

One important task of the prototype design is the division of the RCN into smaller parts, and the application of change detection and change integration to the RCN parts individually. In

order to guide the users through the process effectively, the prototype relies on the functionalities of a *merge sequence generation* module. Its main function is to produce a sequence of focus nodes for a given change target. Hence, the prototype can apply the change detection on each matching pair based on a focus node sequence generated by this module. In addition, this module has functionalities for identifying if the merge process reaches an end. In the following the module functionalities are summarized:

- To generate a sequence of focus nodes for change detection and integration.
- To keep track of the processing states of the nodes.
- To identify the end of the entire merging process.

The merge sequence is based on a breadth-first algorithm. The following list covers the main steps of this module.

- The constructor of the class requires the change target as an input. The *Merge Root* node is selected as the starting point of the main merge process.
- This class can also be processed differently. Provided the user cannot decide to the proposed operation, he can proceed merging with a new starting point of the subtree. In this case, the constructor of the class requires the merge root of the corresponding subtree according to the user selection.
- The first step of the main merge process cycle is to identify a focus node. The module generates a sequence of focus nodes based on the breadth-first algorithm.
  - If all nodes at the current layer have been selected as focus nodes, the prototype collects all children which have not yet been selected in the next tree level, and inserts them into the new layer in the breadth-first container. Then, one node out of this newly-added layer is chosen as the next focus node.
  - If this is not the case, one node which has not been selected as a focus node before in that layer will be selected.
- The entire merge process is considered finished when all of the nodes have been selected into breadth-first container and as focus nodes as well as *processed* by merge algorithms.

In summary, the main algorithm constructing the breadth-first container and selecting a focus node is described below in pseudo code:

Pseudo Code in Constructing Breadth-First Container

```
<Initialization Part>
initialize breadth-first container L
add "Merge Root" or chosen root of subtree in Layer 0 of container L
set index of current layer I with value 0

<Iterative Part>
if (Layer I of container L is not empty) then
    for (each node contained in Layer I of container L) do
        if (that node has not been visited before) then
           set merge flag vector of that node to value "IN_PROCESS"
           return this node
else do
    if (new Layer can be added to container L) then
       add new Layer I+1 to container L
       increase index of current layer I to I+1
       select one node out of this newly-added layer
       set merge flag vector of that node with value "IN_PROCESS"
       return this node
```

3.2.3 Merging Process States

Since the prototype is aimed to provide as much flexibility as possible to the user during change integration, following features are considered.

- The prototype should allow the users to seamlessly switch between change integration mode to navigation mode to inspect the merge candidates.
- The user should be able to select his own focus node and to manipulate the merge sequence. Furthermore, the prototype should also be able to propose an appropriate next focus node.
- Once the user has rejected an operation, the prototype should not propose it again.

In accordance with the mentioned features, six process states (see figure 3.2) are distinguished for the merge process.

- INITIALIZING, this state is entered when the prototype is in the *merge process initialization* step. The prototype expects the user to perform loading merge candidates and selecting merge policy in this phase.
- MATCHING, this state is to execute the match process.
- IN_MERGE_LOOP, when all of the initialization processes are done, and the user has requested to start the change integration, the state is set. The state implies that the prototype is running in the mode of the automatic focus node selection. If the user can then make the change integration decision (*accept* or *reject*) the state changes remain. If the user cannot make the decision, the state changes to RECEIVED_DEFER.

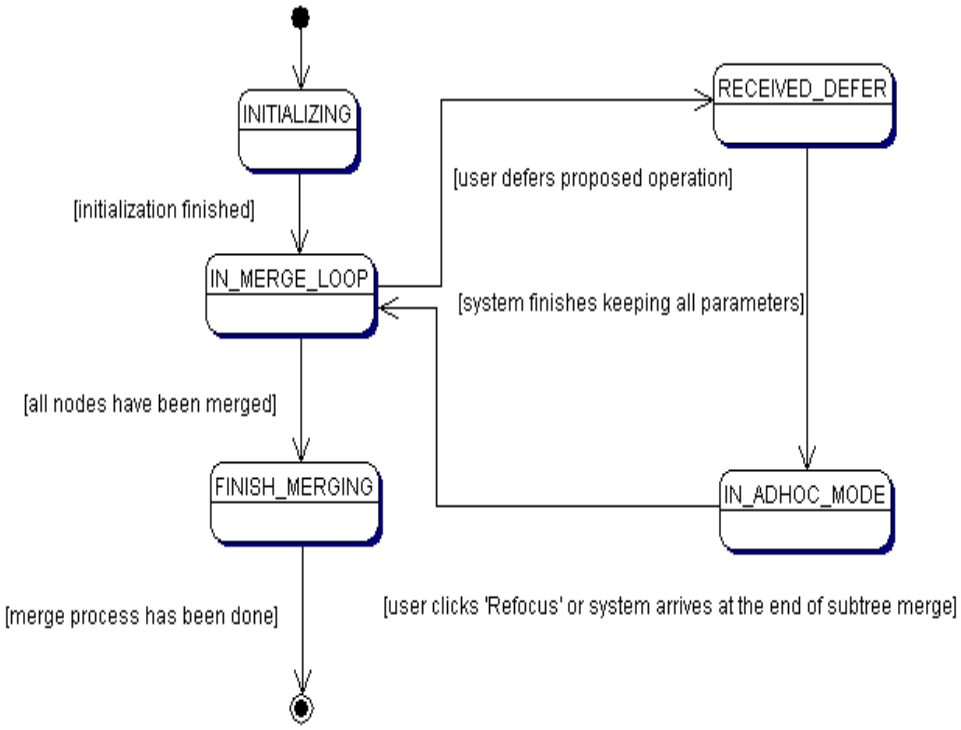

Figure 3.2: State Diagram of Merging Process States

- RECEIVED_DEFER, this state is held whenever the user chooses to postpone the proposed operation. The prototype closes the dialog, keeps the current focus node, changes the state to IN_ADHOC_MODE, and starts to trace the current active node of the change target.
- IN_ADHOC_MODE, the prototype changes to this state automatically from RECEIVED _DEFER. In this state, the user can only navigate content networks and select his own focus node, or resume to the pending process.
  - If the user navigates the content networks, the prototype traces the active node which is requested by the user.
  - If the user selects his own focus node, the prototype resumes the change integration process.
  - If the user is ready to resume the pending process, the state is changed to IN _MERGE_LOOP.
- FINISH_MERGING, the class *merge sequence generation* will determine the entire process termination point, where the prototype state is changed to FINISH_MERGING.

3.3 Change Detection and Change Integration

This section discusses the integration of the process of merge sequence generation, change detection, and change integration. The main consideration is that the change target is changed

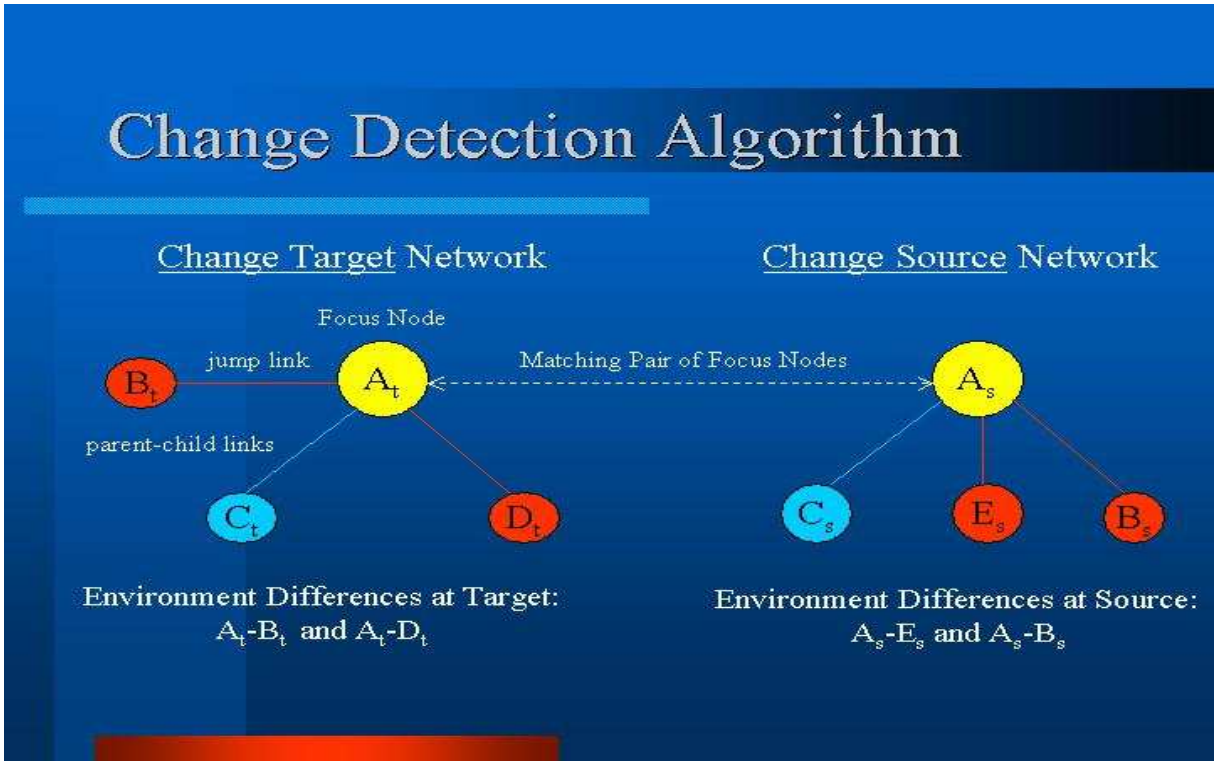

Figure 3.3: Change Detection Process

continuously during the merge process. The prototype is therefore designed to update the internal states of the objects of the merge sequence generation and the change detection after the change integration algorithms are performed.

The process is discussed as follows. All of the objects in the iterations (each of which is composed of *merge sequence generation*, *change detection*, and *change integration*) are initialized at the beginning of the main merge process. In each iteration, the *merge sequence generation* object generates the next focus node (according to the breadth-first algorithm). A corresponding matching pair is then determined. The change detection algorithm is composed of the difference calculation and the situation matching algorithms. The *change integration* algorithm identifies the operations to be performed.

3.3.1 Change Detection Algorithms

This section describes how the prototype detects differences between a given matching pair, and how it maps them to a predefined set of change situations. A change *situation* is an entity that describes a typical way in which the environment may have changed. Each situation is selected based on the results of the difference calculation and is composed of one or more operations. All possible operations are described in the following.

- *Update Content*: Whenever the prototype receives a request from the user to perform the change detection and integration algorithms to a matching node pair, the prototype first decides if there is a necessity to perform the merge of the node content. If this is the case, the *update content* operation is considered to be performed.

Array Name	Description
env1	environments of focus node
env2	environments of focus node matching partner
consens	corresponding links from env1 and env2
delta1	different links from env1 and env2 in change target
delta2	different links from env1 and env2 in change source

Table 3.1: Table of Array Definition in Merge Process

- *Insert Node*: Whenever the prototype does not find a match in the change target to a node in the change source, the *insert node* operation is performed.

- *Insert Link*: Whenever the prototype cannot find a link in the change target that matches one in the change source, provided that both ending nodes of that concerned link did exist in both change target and change source, the *insert link* operation is then performed.

- *Insert both Node and Link*: Whenever the prototype does not find a link in the change target which exists in the change source, and one of the ending node of the link does not exist in the change target, neither, the *insert both node and link* operation is performed.

- *Delete Node*: Whenever the prototype finds that the change source does not contain a node which exists in the change target, the *delete node* operation is performed.

- *Delete Link*: Whenever the prototype finds that the change source does not contain a link which exists in the change target, provided that both of the ending nodes of that concerned link have already existed in both targets, the *delete link* operation is performed.

- *Delete both Node and Link*: Whenever the prototype finds that the change source does not contain a link which exists in the change target, and one of the ending nodes of that link does not exist in the change source, node and link are deleted.

- *Change Type Link*: Whenever the prototype finds that both change source and change target contains two links between two ending matching node pairs, but both link types are different, the prototype will propose a change link type between both ending nodes with respect to the type found in the change source.

- *Redirect Link*: Whenever the prototype finds that there is a link of the same type, one ending node of that link matches to each other, but the other end of each link is connected to the node which does not match to each other (resulted from matching process), the *redirect link* operation is performed.

This section also covers two main issues: the *Difference Computation* algorithm and the *Situation Matching* algorithm.

Number	Situation Name	Description
1	*Update Content*	content or properties of matching pair are different
2	*Change Type Link*	different link types between both matching pair ends
3	*Add Link*	change source contains an additional link
4	*Delete Link*	change target contains an extra link
5	*Add Node and Link*	change source contains an additional link and node
6	*Delete Node and Link*	change target contains an extra link and node
7	*Add Node*	change source contains an additional node
8	*Delete Node*	change target contains an extra node
9	*Redirect Link*	a link in change target has been redirected

Table 3.2: Table of Matching Situations of Focus Node

Difference Calculation Algorithm

This algorithm requires a matching pair as its input parameter. The following operational steps are performed in each iteration:

- The algorithm first determines if the content of two given nodes is different. If the contents are different, update content operation will be considered in the situation matching algorithm.
- For each node, the algorithm keeps all related links in two dedicated arrays, called *env1* and *env2* for change target and change source, respectively.
- The algorithm compares the links in the arrays. If it detects a corresponding pair, that pair will be inserted into the consensus arrays, called *consens1* and *consens2* for change target and change source, respectively.
- The algorithm will then consider only those elements in environment arrays which do not exist in the consensus arrays, and keep those elements in another pair of arrays, called delta arrays, *delta1* and *delta2* for change target and change source, respectively. The two delta arrays are the input for the situation matching algorithm.

Situation Matching Algorithm

As mentioned earlier, this module functions as a transformation module. Given calculated differences are mapped to change situations (see table 3.2). Each is to be performed in the next step. In the following some important scenarios are described:

- *Situation 1*, this situation is considered when contents of the focus node and its matching partner are different.
- *Situation 2*, this situation is considered when there exist two ending nodes of a link in both change target and change source, but the link type found in both candidates are of different types.
- *Situation 3*, this situation is considered when there is a link in the change source, which does not exist in change target.

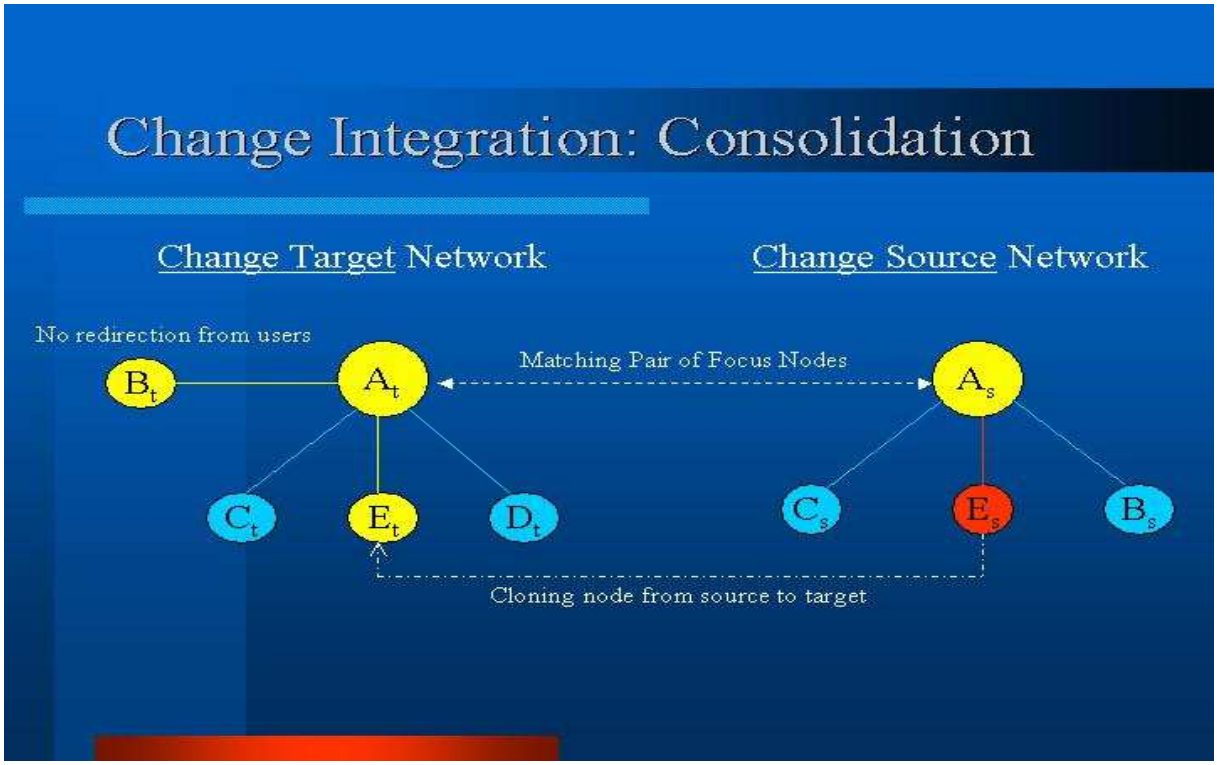

Figure 3.4: Change Integration with Consolidation Merge Policy

- *Situation 4*, this situation is considered when there is a link between two ending nodes which match to each other, while it does not exist in change source.
- *Situation 5*, this situation is considered when there is a link between two ending nodes existing in change source, where the other end of the node and corresponding link both do not exist in change target.
- *Situation 6*, this situation is considered when there is a link between two ending nodes existing in change target, where the other end of the node and corresponding link both do not exist in change source.

Situation Matching algorithm takes the result from *Difference Calculation algorithm* namely, two arrays of *delta1* and *delta2*, and transforms this information to a set of situations for further proceeding. The *Situation Matching algorithm* can be divided into two main parts: checking of potential addition, and checking of potential deletion. The first process phase includes the situations: Situation Add Link, Situation Add Node and Link, and Situation Change Type Link. The second process includes the situations: Situation Delete Link, Situation Delete Node and Link, and Situation Change Type Link.

Checking for Potential Additions

For each link in *delta2*, the algorithm will look through the array *delta1* if it finds a link with corresponding end points.

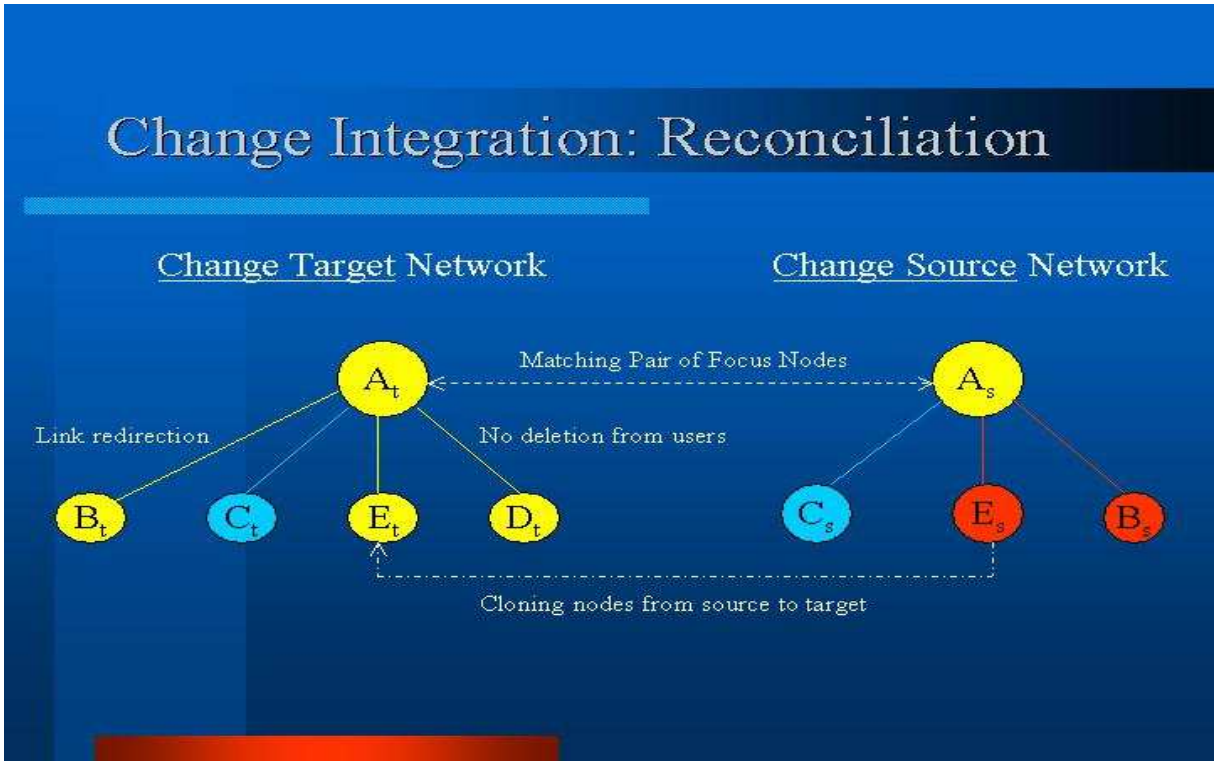

Figure 3.5: Change Integration with Reconciliation Merge Policy

- If it can find such a link in *delta1* and this link is of a different type, the change type link situation will be inserted into the delta collection.
- If it cannot find such a link in *delta1*, it will find matching pair in the entire change target content network. If it exists, the add link situation will be inserted into the delta collection. If there does not exists, the add node and link situation will be inserted into the delta collection instead.

The prototype processes these steps for all elements of the array *delta2*.

Checking for Potential Deletions

For each link in *delta1*, the algorithm will look through the array *delta2* if it finds a link with corresponding end points.

- If it can find such a link in *delta2* and this link is of a different type, the change type link situation will be inserted into the delta collection.
- If no such link exists in *delta2*, it will further find matching pair in the entire change source content network. Accordingly, the delete link situation will be inserted into the delta collection. Or, the delete node and link situation will be inserted into the delta collection instead.

The prototype processes these steps for all elements of the array *delta1*.

Special-Case Situation Matching

However, some special cases are to be considered:

- If a link which is the last link connected to that node is to be deleted, the *Operation Delete Node and Link* will be considered, instead.
- If a last parent link of a node is to be deleted, the *Operation Add Link* between that node to "Merge Root" will be considered additionally.
- If any *Operation Delete Link* or *Delete Node and Link* is to be applied on a node which still has children and/or jumps connected to it and the link which is to be deleted is the last parent link, the prototype will consider the *Operation Delete Link* and add a *Operation Add Link* between that node to "Merge Root" instead.

After all processes in the Situation Matching algorithm are completed, the algorithm will produce an array of situations to be transformed in a sequence of operations afterwards.

3.3.2 Change Integration and Visualization

This step is composed of three main parts, namely Delta Collection, Visualization, and Change Integration. The process requires an input parameter of an array of situations.

Filtering of the Delta Collection

All rejected operations which occurred in earlier processes considering this matching pair will also be taken into account. The overall processes are as follows:

- Retrieve all operations which are rejected by users at earlier processes will be filtered out before being further transfered to the *Change Integration* step.
- Transfer all of those transformed and filtered operation to Change Integration process.

Visualization and Change Integration

All computed operations will be applied from this module. The following items are the corresponding steps:

- Merge policy is set according to the user configuration at the Merge Process Initialization.
- For each operation, the prototype will check the configuration of the merge policy for that particular operation. There are only three possibilities for each operation: apply automatically, interactively, or do not apply at all.
- In case of an interactive type, the effect of the proposed operation is visualized in the change target content network, to support the user decistion.
- If the user rejects the operation, it will be inserted into the array of rejected operations.
- The updated state of the change target will be used in the rest of the merge process. Finally, the integrated change target content network will be stored persistently.

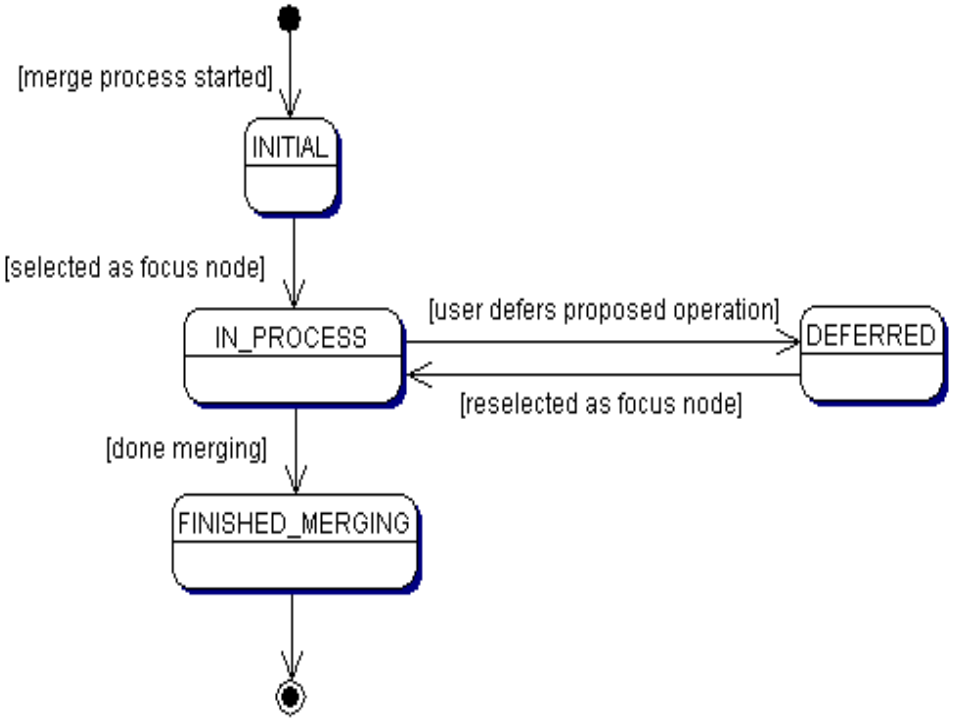

Figure 3.6: State Diagram of Merging States

3.3.3 Merging States

In order to make merge process control more flexible, a merge processing state (see figure 3.6) is managed for each of the nodes in the change target. In searching for the next focus node, the prototype considers all nodes which do not have the merge flag set to FINISH_MERGING. While the final point of the merge process will be determined only if all nodes in change target have merge flag set to FINISH_MERGING. All possible states are itemized as follows:

- INITIAL, all nodes which have never been selected as focus nodes will hold this state.
- IN_PROCESS, for each node which is selected to be the focus node will be set to this state. After processing with the change detection and change integration, the node is set its state with either DEFERRED or FINISH_MERGING.
- DEFERRED, whenever an operation is applied interactively, the user can defer that operation. Consequently, that node will be set to DEFERRED, and will be selected as a focus node again before the merge process finishes.
- FINISH_MERGING, this state will be held when the merging process of a given matching pair is finished without any deferring request from the user.
- CHANGE_DETECTED, this state is held when the user has changed some part of the change target content network during the merging process directly. However, the prototype does not include this feature. It could be considered in the future work.

3.3.4 Merge Session

The merge session feature gives the user a possibility to postpone the merge session and resume the pending process at the saved point. It poses additional requirements to the prototype. It should be able to keep all important information about the current processing state persistently. To achieve this idea, the following data members are of importance:

- All merge candidates, especially meaning change target and change source,
- chosen merge policy,
- latest matching result set,
- the merge states of the nodes, and,
- all rejected operations.

However, due to the limitation of the complexity of the prototype design, this prototype version considers only the storage of the merge candidates and the merge policy. It leaves all other items for the future implementation.

Chapter 4

Employed Technologies

In the first three chapters the concepts of the prototype design process are explained. Before the prototype functions and the graphic user interfaces (GUI) issues are discussed, the author discusses two technologies which are used in prototype implementation. They are the object-oriented programming language Java and the employed RCN format.

4.1 Java

Concerning the problem to construct the software application which merges resource-classifying networks (RCNs), the author chooses *Java*, the well-known object-oriented programming language, according to the following reasons.

- *Java* has the following key advantages of an object-oriented programming language:
  - It couples characteristics and behaviors into objects.
  - It enables information hiding.
  - Developed classes can be easily reused by *Inheritance* and *Composition*.
  - *Polymorphisms* support a high flexibility of the implementation extensibility.
- Java is platform-independent.
- Java has a comprehensive library collection, which is helpful in creating any complex programs.
- Java has powerful GUI libraries.

4.2 TheBrain

4.2.1 RCN Format Requirements

Regarding the project requirements to create the prototype for merging RCNs, any participating RCN should conform with the following important requirements:

- The chosen RCN format support classifier nodes, content nodes, parent-child links, and jump links,

Figure 4.1: An Example of Resource-Classifying Network: TheBrain

- the chosen RCN should be able to be modified by the chosen programming language, in our case Java language, via existing application programming interfaces (APIs),
- the chosen RCN should be able to be effectively visualized, and,
- the chosen RCN should be able to organize the various types of information resources.

4.2.2 The Brain RCN format

The prototype depends on the BrainSDK packages, which provides an application programming interfaces (APIs) to construct and modify the *brain* objects. The packages are composed of the database, the user-interface and the common module. The database package has three primary functions:

- The persistent storage and retrieval of the *brain*,
- The creation, the association, and the access of the thought objects, and
- The associating of *thoughts* and *links*.

The user-interface package manages the visual display and communicates with the database package to access the *thoughts* and *links*. The prototype main algorithm is based on the BrainSDK database package, while the prototype GUI is based on the BrainSDK user-interface package.

TheBrain uses the following terminologies to describe a RCN, for instance, The *brain* is a collection of the *thoughts* and the associations between those thoughts. The *thoughts* are the basic building blocks. There are four roles for thought with respect to a given thought.

- The *Parent Thought* of a *thought* represents a super-category of a clarifying concept of that thought.

- The *Child Thought* is the reverse role of the *parent thought* A *parent thought* may have many child thoughts.
- The *Jump Thought* links different groups among the families of *thoughts*. It represents a loose association between any two nodes.
- The *Sibling Thoughts* are the other children which share the same *parent thought*.

These thought roles are supported by the following three types of *links*:

- The *Parent Link* connects a *thought* to a *parent thought*.
- The *Child Link*, a *parent thought* may have many *child thoughts*.
- The *Jump Link* connects the different groups or families of the separate, but related, thoughts.

4.2.3 The BrainSDK

Evaluating these requirements, the RCN format *TheBrain* [3] has been chosen for the following reasons.

- *TheBrain* has the strong advantage of an effective visualization support.
- *TheBrain* can implement the classifier nodes and the content nodes with the *thoughts* and it implements the parent-child links with *links*.
- *TheBrain* format comes with Java libraries for its storage and visualization.
- *TheBrain* can organize a wide variety of information resource because it refers to locations via URLs, which can refer both local and global resources.

Chapter 5

Prototype Functions and Graphical User Interfaces

In the early chapters of this report, the author explains why the prototype is constructed, what the aims of the prototype are, how the prototype is designed, and which technologies the prototype employs and depends on. This chapter explains implementation issues of the prototype, its architecture, and the graphical user interface.

5.1 Prototype Functions

The prototype is constructed to realize the merge functionality for an available RCN format. This prototype version is limited to two merge candidate–a change target and a change source–because configurations with more than two candidates are technically feasible but would overstrain the user ability. However, the prototype could be improved to implement three- or more-way merge in the future. In summary, the following items are the functions of this prototype version:

- capable of performing matching with adaption through the user,
- capable of performing automatic change detection, which is based on the content and the enviroments of each node pair.
- capable of performing automatic, interactive, or mix-type change integration.
- capable of handling arbitrary navigation in the networks anytime during the merge process.
- capable of providing the user with the merge policy customization and selection.

5.2 Prototype Architecture

The prototype is composed of four main components (see figure 5.1):

- 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 merge policy.

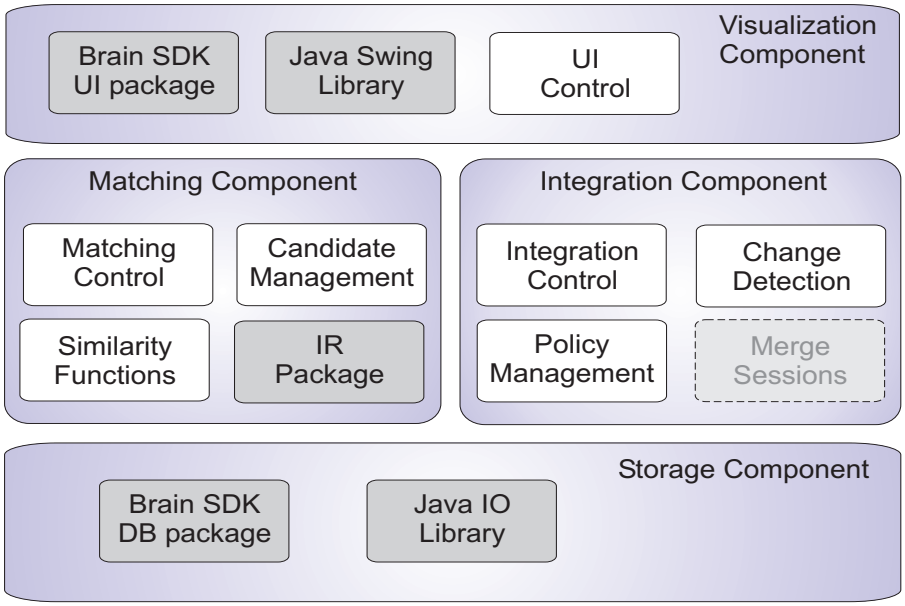

Figure 5.1: Prototype Architecture

- The **matching component** provides the functionality for the computation of the RCN matching. It includes a component for the management of the matching candidates, matching pairs and single lists as well as a component for the control of the matching process.

- 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 which provides functionality for the persistent storage of the employed RCN format in the file system. For the storage of further merge-related information like user-defined merge policies the functionality of the Java IO library is used.

5.3 User Interface and User Interaction

5.3.1 User Interface Overview

The prototype visualization and GUI component (see figure 5.2) depends on the JFC Swing packages. The handling of user interaction is based on an event-driven implicit invocation style. Each component is registered with the events of the user interface, and this causes an invocation of the registered procedure when the considered events are accessed at run-time [6]. The main registered modules of the prototype are

- Two panels of the merge candidates, the left-hand-side panel has the role of the change target, while right-hand-side one has the role of the change source.

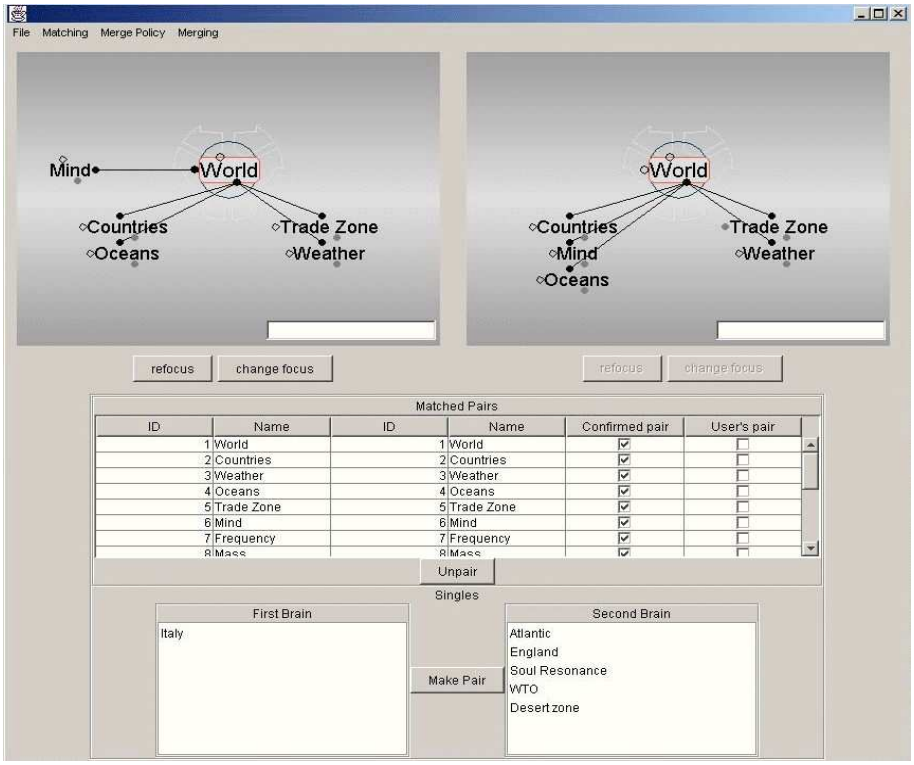

Figure 5.2: Prototype User Interface Overview

- The matching result set panel shows a set of the matching results. The prototype triggers the new corresponding active node when user clicks each element in the table.
- The menu-bar which is composed of the merge and the match configurations, the merge policy customization and selection, and the merge process execution and reinitialization.

5.3.2 User Interaction and User Intervention

The prototype assumes the user interaction as one of the crucial requirements, the author depicts the user interaction issues into two following parts.

- *User interactions in the merge iterations*, once the merge configurations are identified (loading two merge candidates, executing the matching process, and identifying the merge policy), the merge process is triggered to start. The user is now considered in the merge iterations. The prototype automatically selects the focus node, computes the change detection algorithms, and interacts with the user in case the detected change operation is configured to do so. To achieve the benefit of the automatic change detection and integration, the prototype is constructed to limit user interaction in this case. As long as all of the operations are configured to be integrated automatically, the users have no possibilities to interrupt the running process.
- *User interactions out of the merge iterations*, despite the strict interaction limitations in the first mode according to the merge policy configuration, the prototype is also set up to run iteratively. In this mode, the user has the freedom to arbitrarily navigate the

most-updated integrated merge candidates. The user enters this mode by two following ways:

- when the merge process has yet been triggered to start, the user is expected to identify the merge configuration.
- when the merge process has already started, but the entire merge process has not finished yet. The user enters this mode whenever the user is not ready to make the decision on any integrated operation. He refuses to make the decision. After that point, the prototype will run passively, the user can choose three following possibilities to resume the main merge process.
  * The user can select the new starting node for the new temporary merge process,
  * The user can select to resume to the main merge process where he postponed it earlier, or,
  * The user can select to restart the overall process again by reinitializing the entire merge process.

5.3.3 Visualization of Proposed Change

The prototype always performs the automatic differencing calculations, and *proposes the user with the questions whether he agrees to integrate the operation* (see figure 5.3) to the change target or not. However, if the operation has other two values, according to the merge matrix element value:

- if it has the value of IGNORED, the prototype ignores the operation.
- if it has the value of AUTOMATIC, the pprototype integrates the operation.

According to this fact, we can consider that the tight couple relationship among the *Change Detection*, the *Change Integration*, and the *Merge Matrix Configuration* module can yield the compact and effective behavior as aimed.

Prototype Recommendations

The prototype automatically integrates or ignores the operation if that operation is not configured in the merge matrix with the value of USER, otherwise, it proposes the user with the question if the prototype should integrate that operation. There are three possibilities for user:

- If user chooses YES, the operation is performed.
- If user chooses NO, the operation is ignored.
- If users chooses DEFER, the operation is then ignored, and it will postpone the running iteration and change the mode to be passive. At this point, the user has a chance to navigate through the contexts of both merge candidates. Once he is ready to continue the process, he may select to start with following options:
  - to select the focus node by himself, and run the temporary new merge sequence until the end of that subtree (please go to the next subsection *Manual Merge Sequence Selection–Subtree Merge* for more details), or,

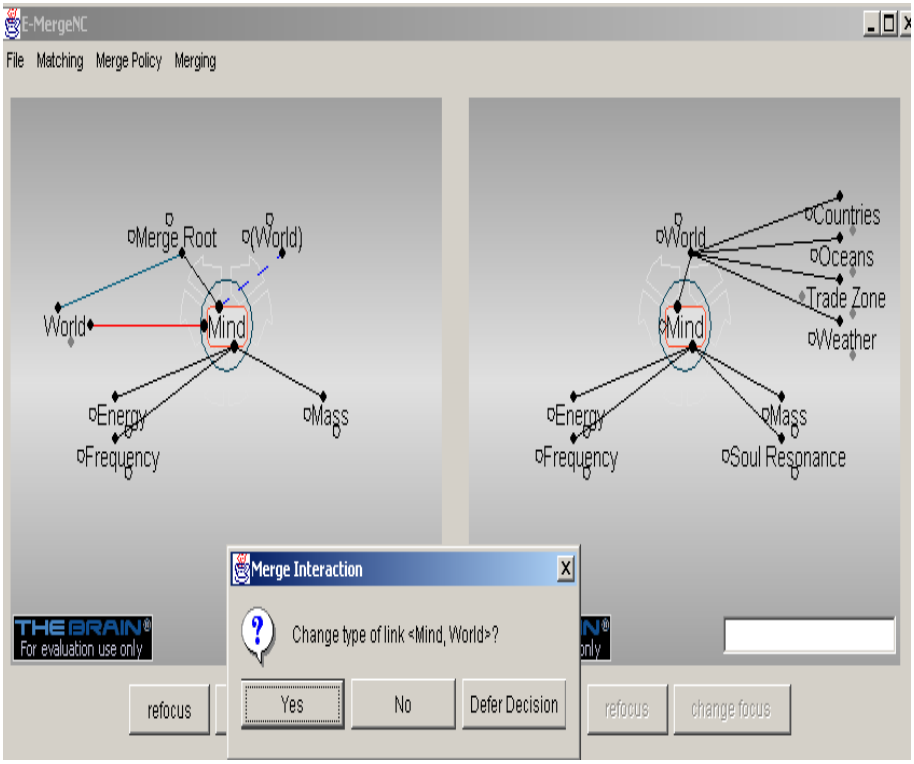

Figure 5.3: Visualization of Proposed Change

– to select the focus node by himself, and run the temporary new merge process at the given node, as well as resume the pending process after the prototype finishes the new merge process (please go to the next subsection *Manual Merge Sequence Selection–Single-Node Merge* for more details), or,

– to resume the pending process, since he is ready to further proceed with pending questions. (please go to the next subsection *Automatic Merge Sequence Resumption* for more details)

Manual Merge Sequence Selection

The user arrives at the situation when he has deferred an operation earlier, and the operation is still pending. This module is designed to give user the freedom in proceeding with the new arbitrary merge on other parts of the change target. As soon as the user defers a operation, the prototype starts to trace the user interaction with the change target, and also keeps updating the current active node. Immediately after the user clicks CHANGE FOCUS, the mode *Manual Merge Sequence Selection* is assumed. There are two options once user enters this module.

- The *Subtree Merge*: The user may start to run the new temporary merge process for the whole subtree with the current active node as the *subtree root*. The prototype assumes to start the mode *Automatic Merge Sequence Resumption* once the temporary merge process has done.

- The *Single-Node Merge*: The user may start to run the new temporary merge process for a single-node merge at the current active node. The prototype assumes to start the *Automatic Merge Sequence Resumption* once the temporary merge process has done.

Automatic Resumption

The user arrives at the situation when he has deferred an operation earlier, and the operation is still pending. The module is designed to enable the user to go back to the point where he deferred earlier. Once the module is triggered to run, the prototype starts running the main iteration of the change detection and the change integration from the point where he left before. The main merge process finishes when the prototype has performed merging on all nodes of the merge candidates. Additionally, all of the proposed operations which the user has decided to refuse will not be proposed again.

5.3.4 Merge Policy Characteristics

The merge policy employed by the prototype is based on the purpose to solve the possible conflicts. The merge policy can also give user the possibility to specify at which situations he prefers the prototype to prompt the questions. In our situation, eventhough the prototype limits to accept only to two merge candidates, the prototype has already provided a merge policy customization implementation, so-called the *Merge Matrix*.

Merge Matrix

The implemented merge matrix has two dimensions, each of which depends on the numbers of the possible integrated operations. Each dimension represents the computed operations based on the results of the change detection process. However, in the prototype there are only two merge candidates, therefore, we are interested only in one dimension (row), while the other dimension (column) is always considered with the value *no operation*.

Level of Automation and Interactivity

The entries of the matrix specify the action which the merge procedure should operate on the merge target. There are three possible values:

- AUTOMATIC refers to the automatic integration.
- IGNORED refers to ignore the operation.
- USER refers to the integration with this computed operation interactively with user.

Chapter 6

Conclusion

The prototype consists of two main parts, namely the matching and the integration component, and the two components depend on each other. The integration of both modules realizes the full functionalities of the merge prototype. However, the report mainly focuses on the latter part, the integration component. Meanwhile, the other part has conducted by another project in parallel. Nevertheless, the author concludes the prototype construction with following issues by focusing on the merge process as a whole.

- The prototype is constructed to provide the merge functionality especially for the resource-classifying network information type.
- The prototype is aimed at the semi-automatic mode process. It also provides the user with the merge policy customization feature.
- The prototype is adapted to perform the merge algorithms on the resource-classifying networks, which focus on the operations to the environments of the node.
- The merge process consists of identifying the merge configuration, the matching, a set of the change detection, the change integration, and the visualization modules.
- The prototype is also aimed at further integration to the larger framework.

Furthermore, the merge tool prototype can be extended in several directions. To merge larger RCNs it is desirable to have persistent merge sessions that can be interrupted and resumed later without losing the effort invested. The prototype should also be extended to support more complex change operations and adequate visualizations for them. Additionally the author intend to examine options for three network merge configurations to perform a three-way merge equipped with an effective conflict management.

Bibliography

[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, November 3-5, 1993*, Lecture Notes in Computer Science, pages 249–257. ACM, November 1993.

[2] Hendry Chandra, Ulrike Steffens, and Claudia Niederée. Semi-automatic Merging of Content Networks: Customized Matching. April 2001.

[3] TheBrain Technologies Corporation. TheBrain, http://www.thebrain.com/, February 2001.

[4] D. MacKenzie, P. Eggert, and R. Stallman. Comparing and Merging Files: diff, diff3, sdiff, cmp, and patch, Edition 1.3. September 1993.

[5] J. Munson and P. Dewan. A Flexible Object Merging Framework. In *Proceedings of the ACM CSCW'94 Conference on Computer Supported Cooperative Work, October 1994, Chapel Hill, NC*, pages 231 – 242. ACM-Press, October 1994.

[6] Marry Shaw and David Garlan. *Software Architecture*. Prentice-Hall, Inc., 1996.

[7] Hull R. Widjojo, S. and Wile. D. A Specificational Approach to Merging Persistent Object Bases. *The International Workshop on Persistent Object Systems*, pages 267 – 278, September 1990.